前言　MATLAB 6.5

关于本丛书

随着计算机性能的不断提高，人们发现工程上的许多问题可以通过计算机强大的计算功能来辅助完成。如此一来，MATLAB 软件强大的数值运算核心开始被关注。经过近 20 年的发展，MATLAB 的核心被进一步完善和强化，同时许多工程领域的专业人员也开始用 MATLAB 构造本领域的专门辅助工具，这些工具后来便发展为 MATLAB 的各种工具箱。特别值得一提的是，MATLAB 是一种开放式的软件，任何人通过一定的程序都可以将自己开发的优秀的应用程序集加入到 MATLAB 工具的行列中。这样，许多领域前沿的研究者和科学家都可以将自己的成果集成到 MATLAB 之中，被全人类继承和利用。因此，我们现在看到的 MATLAB 才会如此强大和丰富。

MATLAB 强大的功能在于它的开放式的可扩展环境及诸多的面向不同应用领域的工具箱（Toolbox），包括信号处理工具箱（signal processing toolbox）、图像处理工具箱（image processing toolbox）、通信工具箱（communications toolbox）、小波分析工具箱（wavelet toolbox）、系统辨识工具箱（system identification）、优化工具箱（optimization toolbox）、控制系统工具箱（control system toolbox）、鲁棒控制工具箱（robust control toolbox）、非线性控制工具箱（nonlinear control toolbox）等，而且工具箱现在还在不断地扩展。

然而，目前国内图书市场上有关 MATLAB 方面的书籍要么侧重于 MATLAB 语言编程介绍，要么侧重于各种工具箱函数的解说，而对怎样应用工具箱函数来解决实际问题鲜有涉及，致使大多数读者面临熟悉 MATLAB 工具箱函数，但不会用来解决实际问题的尴尬局面。针对这种现状，电子工业出版社计算机研发部推出了《MATLAB 应用技术》丛书，希望能为读者在 MATLAB 软件工具箱和工程实际问题之间架起一座桥梁，帮助读者提高使用 MATLAB 工具箱分析问题、解决问题的能力。

本丛书主要面向工程应用及实际问题的解决。通过大量的示例和具体应用实例来讲解一些复杂理论的应用，使读者不必详细了解这些复杂的理论而能够使用 MATLAB 提供的方法在工程实际中使用和实现那些复杂的理论。本从书试图通过简单快速的方法使读者能够通过运用 MATLAB 来解决工程上的实际问题，避免去深究一些复杂的具体理论，通过对大量的实例分析和说明让读者掌握如何去使用 MATLAB 这门辅助工程设计工具，以及在具体问题中怎样来使用 MATLAB 进行分析和辅助设计。

丛书侧重于 MATLAB 在工程领域的具体应用，涉及了科学计算、数字信号处理、小波分析、神经网络、图像处理、MATLAB 与 VC 接口等领域，涵盖面之广，可谓空前。

本丛书的编写宗旨是：充分、详尽地介绍当今 MATLAB 最新版本（即 6.5 版）的各种通用性功能。各章节在介绍 MATLAB 各种指令综合运用的同时都尽量加入了不少的例子，较好地弥补了 MATLAB 用户指南和在线帮助的不足。

书中对于 MATLAB 基本编程语言及一些矩阵运算的知识基本不予介绍，读者在使用本书前应当有最基本的 MATLAB 矩阵运算和简单编程语句的知识，以及一些相关专业的基本知识。

关于本书

本书由浅入深地介绍了 MATLAB 6.5 图像处理工具箱 v3.2 的一些基本概念，比如图形句柄、图像的表示等，及 MATLAB 在各种图像处理中的应用，例如图像变换、设计 FIR 滤波器、图像增强、四叉树分解、边缘检测、二值图像的操作，以及小波分析和分形几何用于图像处理的 MATLAB 实现等，涵盖了我们在工程实践中经常遇到的一些问题。为了让读者对这些方法的使用有更进一步的认识，也适当地穿插了一些理论方面的介绍。最后的提高部分给出了一些图像处理技术综合应用的实现。一个优秀图像处理算法的演示往往离不开一个友好的图形操作界面，从工程角度出发，我们在最后介绍了图形界面设计（GUI）方面的内容。本书内容涉及到了 MATLAB 在图像处理中的大部分应用，希望能够帮助用户朋友解决一些实际问题。（为了查阅方便，本书最后给出了具有重要参考价值的附录。书中实例的源代码可至飞思在线 http://www.fecit.com.cn 下载。）

本书由飞思科技产品研发中心策划并组织编写，本书第一部分由陈荣娟执笔，第二部分由郭颖辉执笔，第三部分由陈杨执笔。全书由陈杨校正，另外还有很多同志在本书的排版、校对过程中付出了大量的劳动，在此一并致谢！

由于时间仓促及作者本身水平有限，书中错误之处在所难免，敬请各位专家和广大读者批评指正！

我们的联系方式：

电　　话：（010）68134545　　68131648

电子邮件：support@fecit.com.cn

飞思在线：http://www.fecit.com.cn　　http://www.fecit.net

答疑网址：http://www.fecit.com.cn/question.htm

通用网址：计算机图书、飞思、飞思教育、飞思科技、FECIT

飞思科技产品研发中心

目录

MATLAB 6.5

第 1 章 MATLAB 句柄图形对象

句柄图形（Handle Graphics）是 MATLAB 6.5（以下简称 MATLAB）提供给用户的一种用于创建计算机图形的面向对象的图形系统。图形句柄提供了多种绘图命令用于创建线条、文本、网格和多边形等，以及交互式图形界面，如菜单、按钮和对话框等。利用图形句柄，用户可以直接操纵线条、表面和其他图形对象。MATLAB 中的绝大多数高级例程（即高级图形函数和高级绘图命令）正是利用这些图形元素产生各种类型的图形。利用图形句柄，我们可以在 MATLAB 的命令行中修改图形的显示效果，也可以在 M 文件中创建用户自定义的绘图函数。

本章主要内容：

- MATLAB 6.5 语言简介
- MATLAB 的图形对象及简单应用
- 图形对象的属性设置与查询
- 图形对象句柄的访问

1.1 MATLAB 6.5 语言简介

MATLAB 语言是一种非常强大的工程语言，本节主要对该语言进行简单的介绍，包括 MATLAB 的产生背景及主要产品、MATLAB 的语言特点及 MATLAB 6.5 新版本所具有的新特点。

1.1.1 MATLAB 的产生背景及主要产品

MATLAB 诞生在 20 世纪 70 年代，它的编写者是 Cleve Moler 博士和他的同事。当时，Cleve Moler 博士和他的同事开发了 EISPACK 和 LINPACK 的 FORTRAN 子程序库。这两个程序库主要是求解线性方程的程序库。但是，Cleve Moler 发现学生使用这两个程序库时有困难，主要是接口程序不好写，很费时间，于是 Cleve Moler 自己动手，在业余时间里编写了 EISPACK 和 LINPACK 的接口程序。Cleve Moler 给这个接口程序取名为 MATLAB，意为矩阵（MATRIX）和实验室（LABORATORY）的组合。以后几年，MATLAB 作为免费软件在大学里使用，深受大学生的喜爱。

1984 年，Cleve Moler 和 John Little 成立了 MathWorks 公司，正式把 MATLAB 推向市场，并继续进行 MATLAB 的开发。1993 年，MathWorks 公司推出 MATLAB 4.0；1995 年，MathWorks 公司推出 MATLAB 4.2C 版（For Win3.x）；1997 年推出 MATLAB 5.0；2000 年 10 月，MathWorks 公司推出 MATLAB 6.0；2002 年 8 月，新的版本 MATLAB 6.5 已经开始发布了。每一次版本的推出都使 MATLAB 有了长足的进步，界面越来越友好，内容越来越丰富，功能越来越强大。它的帮助信息采用超文本格式和 PDF 格式，可以很方便地阅读。

MATLAB 长于数值计算，能处理大量的数据，而且效率比较高。MathWorks 公司在此基础上开发了符号计算、文字处理、可视化建模和实时控制能力，增强了 MATLAB 的市场竞争力，使 MATLAB 成为了市场主流的数值计算软件。

MATLAB 产品族支持从概念设计、算法开发、建模仿真、实时实现的理想的集成环境。无论是进行科学研究还是产品开发，MATLAB 产品族都是必不可少的工具。

MATLAB 产品族可用来进行：

- 数据分析
- 数值和符号计算
- 工程与科学绘图
- 控制系统设计
- 数字图像信号处理
- 财务工程
- 建模、仿真、原型开发
- 应用开发
- 图形用户界面设计

MATLAB 产品族被广泛地应用于包括信号与图像处理、控制系统设计、通信、系统仿真等诸多领域。开放式的结构使 MATLAB 产品族很容易针对特定的需求进行扩充，从而在不断深化对问题的认识同时，提高自身的竞争力。

MATLAB 产品族的一大特性是有众多的面向具体应用的工具箱和仿真块，包含了完整的函数集用来对信号图像处理、控制系统设计、神经网络等特殊应用进行分析和设计。其他的产品延伸了 MATLAB 的能力，包括数据采集、报告生成和依靠 MATLAB 语言编程产生独立的 C/C++代码等。

MATLAB 主要产品构成如下：

- MATLAB

所有 MathWorks 公司产品的数值分析和图形基础环境。MATLAB 将 2D 和 3D 图形、MATLAB 语言能力集成到一个单一的、易学易用的环境之中。

- MATLAB Toolbox

一系列专用的 MATLAB 函数库，解决特定领域的问题。工具箱是开放的可扩展的，可以查看其中的算法，或开发自己的算法。

- MATLAB Compiler

将 MATLAB 语言编写的 M 文件自动转换成 C 或 C++文件，支持用户进行独立应用开发。结合 MathWorks 提供的 C/C++数学库和图形库，用户可以利用 MATLAB 快速地开发出功能强大的独立应用。

- Simulink

是结合了框图界面和交互仿真能力的非线性动态系统仿真工具。它以 MATLAB 的核心数学、图形和语言为基础。

- Stateflow

与 Simulink 框图模型相结合，描述复杂事件驱动系统的逻辑行为，驱动系统可以在不同的模式之间进行切换。

- Real-Time Workshop

直接从 Simulink 框图自动生成 C 或 Ada 代码，用于快速原型和硬件在回路仿真，整个代码生成可以根据需要完全定制。

- Simulink Blockset

专门为特定领域设计的 Simulink 功能块的集合，用户也可以利用已有的块或自编写的 C 和 MATLAB 程序建立自己的块。

1.1.2　MATLAB 语言特点

MATLAB 语言有不同于其他高级语言的特点，它被称为第四代计算机语言。正如第三代计算机语言如 FORTRAN 语言与 C 语言等使人们摆脱了对计算机硬件的操作一样，MATLAB 语言使人们从繁琐的程序代码中解放出来。它的丰富的函数使开发者无需重复编程，只要简单地调用和使用即可。MATLAB 语言最大的特点是简单和直接。MATLAB 语言的主要特点有：

1．编程效率高

MATLAB 是一种面向科学与工程计算的高级语言，允许用数学形式的语言编写程序，且比 BASIC、FORTRAN 和 C 等语言更加接近我们书写计算公式的思维方式，用 MATLAB 编写程序犹如在演算纸上排列公式与求解问题，因此，也可通俗地称 MATLAB 语言为演算纸式科学算法语言。由于它编写简单，所以编程效率高，易学易懂。

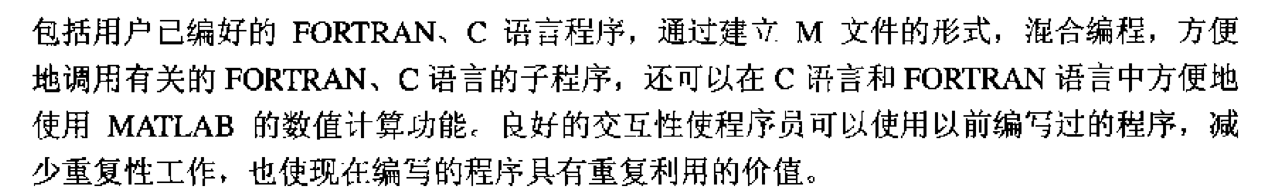
2．用户使用方便

MATLAB 语言是一种解释执行的语言（在没被专门的工具编译之前），它灵活、方便，其调试程序手段丰富，调试速度快，需要学习时间少。人们用任何一种语言编写程序和调试程序一般都要经过四个步骤：编辑、编译、连接，以及执行和调试。各个步骤之间是顺序关系，编程的过程就是在它们之间做瀑布型的循环。MATLAB 语言与其他语言相比，较好地解决了上述问题，把编辑、编译、连接和执行融为一体。它能在同一画面上进行灵活操作，快速排除输入程序中的书写错误、语法错误甚至语意错误，从而加快了用户编写、修改和调试程序的速度，可以说在编程和调试过程中它是一种比 VB 还要简单的语言。

具体地说，MATLAB 运行时，如直接在命令行输入 MATLAB 语句（命令），包括调用 M 文件的语句，每输入一条语句，就立即对其进行处理，完成编译、连接和运行的全过程。又如，将 MATLAB 源程序编辑为 M 文件，由于 MATLAB 磁盘文件也是 M 文件，所以编辑后的源文件就可直接运行，而不需进行编译和连接。在运行 M 文件时，如果有错，计算机屏幕上会给出详细的出错信息，用户经修改后再执行，直到正确为止。所以可以说，MATLAB 语言不仅是一种语言，广义上讲更是一种语言开发系统，即语言调试系统。

3．扩充能力强，交互性好

高版本的 MATLAB 语言有丰富的库函数，在进行复杂的数学运算时可以直接调用，而且 MATLAB 的库函数同用户文件在形式上一样，所以用户文件也可作为 MATLAB 的库函数来调用。因而，用户可以根据自己的需要方便地建立和扩充新的库函数，提高 MATLAB 使用效率和扩充它的功能。另外，为了充分利用 FORTRAN、C 等语言的资源，



包括用户已编好的 FORTRAN、C 语言程序，通过建立 M 文件的形式，混合编程，方便地调用有关的 FORTRAN、C 语言的子程序，还可以在 C 语言和 FORTRAN 语言中方便地使用 MATLAB 的数值计算功能。良好的交互性使程序员可以使用以前编写过的程序，减少重复性工作，也使现在编写的程序具有重复利用的价值。

4．移植性好，开放性好

MATLAB 是用 C 语言编写的，而 C 语言的可移植性很好。于是 MATLAB 可以很方便地移植到能运行 C 语言的操作平台上。MATLAB 适合的工作平台有：Windows 系列、UNIX、Linux、VMS 6.1、PowerMac。除了内部函数外，MATLAB 所有的核心文件和工具箱文件都是公开的，都是可读可写的源文件，用户可以通过对源文件的修改自己编程构成新的工具箱。

5．语句简单，内涵丰富

MATLAB 语言中最基本最重要的成分是函数，其一般形式为［*a*，*b*，*c*，…］= *fun*（*d*,*e*,*f*,…），即一个函数由函数名，输入变量 *d*,*e*,*f*, …和输出变量 *a*,*b*,*c*，…组成。同一函数名 *fun*，不同数目的输入变量（包括无输入变量）及不同数目的输出变量，代表着不同的含义（有点像面向对象中的多态性）。这不仅使 MATLAB 的库函数功能更丰富，而且大大减少了需要的磁盘空间，使得 MATLAB 编写的 M 文件简单、短小而高效。

6．高效方便的矩阵和数组运算

MATLAB 语言像 BASIC、FORTRAN 和 C 语言一样规定了矩阵的算术运算符、关系运算符、逻辑运算符、条件运算符及赋值运算符，而且这些运算符大部分可以毫无改变地照搬到数组间的运算中，有些如算术运算符只要增加“·”就可用于数组间的运算。另外，它不需定义数组的维数，并给出矩阵函数、特殊矩阵专门的库函数，使之在求解诸如信号处理、建模、系统识别、控制、优化等领域的问题时，显得大为简捷、高效、方便，这是其他高级语言所不能相比的。在此基础上，高版本的 MATLAB 已逐步扩展到科学及工程计算的其他领域。因此，不久的将来，它一定能名副其实地成为“万能演算纸式的”科学算法语言。

7．方便的绘图功能

MATLAB 的绘图功能是十分方便的，它有一系列绘图函数（命令），例如线性坐标、对数坐标、半对数坐标及极坐标。只需调用不同的绘图函数（命令），即可在图上标出图题、*XY* 轴标注，格（栅）绘制也只需调用相应的命令，简单易行。另外，在调用绘图函数时调整自变量可绘出不变颜色的点、线、复线或多重线。这种为科学研究着想的设计是通用的编程语言所不及的。

1.1.3 MATLAB 6.5 新特点

2002 年，MathWorks 公司发布 MATLAB 最新产品 6.5 版。MATLAB 6.5 的特点在于全新的桌面及各种不同领域的集成工具，使用户易于使用。多种新工具简化了一般的工作，如资料输入、快速分析，并可创造出高品质且具实用性的图表分析等。MATLAB 6.5 包含

了新的 JIT 加速度计。JIT 加速度计极大地加快了 MATLAB 中的许多操作的计算速度。其他新的特色包括以下三个方面：

1．编程和数据类型

1）增加了变量名、函数名和文件名的最大长度：变量名、函数名、子函数名、结构域名、M 文件名、MEX 文件名和 MDL 文件名可以达到 63 个字节。

2）支持 64 位的文件偏移量，能够为大于 2GB 的数据文件实现低层次的 I/O 函数。

3）支持有符号和无符号的 64 位整数。

4）支持使用动态域名来访问和修改结构数组。

5）简化了 AND 和 OR 逻辑运算。

6）支持新的 MATLAB 定时器，而不是定时执行 MATLAB 命令。

7）改进了音频支持。

8）加强了警告和错误提示功能，支持格式化的字符串和消息标识符。

2．外部接口

1）改进了自动化客户接口：新的查看和修改属性用户接口，增强了事件和例外句柄。

2）增强了网络集成：读 URL 的内容，在 MATLAB 中发送 E-mail 及解压缩文件。

3．开发环境

1）新的 M-文件接口，能更好地理解 M-代码

2）新的启动按钮，易于执行共同的命令。

3）改进了文件和目录管理工具。

4）增强了数组编辑器：与 Excel 之间剪切、复制、删除和交换单元的新功能，支持更大的数组。

5）改进了编辑和调试工具。

6）改进与 PC 平台的控制接口。

7）支持新的图形用户接口，可以从 HDF 或 HDF-EOS 文件导入数据。

8）JVM 1.3.1 支持 Windows，Linux 和 Solaris 平台。

4．图形

提高了图形性能：新的彩色图形编辑器，改进了图形特性编辑器。

5．数学

1）新的数学计算和算法改进。

2）在 Pentium4 上更快地计算许多函数，比如矩阵乘法、矩阵转置和线性代数运算。

在这里应特别指出的是，许多新特色并不是适合于所有的平台。

1.2　MATLAB 的图形对象及简单应用

图形句柄对象是 MATLAB 用来显示数据和创建图形用户界面（GUI）的基本绘图元素。对象的每个实例（instance）均对应于一个惟一的标识符，即句柄（handle）。*利用该*

句柄，用户就可以非常容易地操纵现有的图形对象的各种特征（即对象属性）。MATLAB 中，对象的从属关系如图 1-1 所示。

从图中可以看出，所有 MATLAB 的图形对象都是彼此联系的。因此，典型的 MATLAB 图形显示均包含多种对象，而不是单一的。

MATLAB 遵循面向对象的原则，每个图形对象都有一个相应的构造函数，用于创建该对象类的实例。对象构造函数的名称与其构造的对象名相同。例如，text 函数创建 Text 对象（即文本对象）的实例，figure 函数创建 Figure 对象的实例，等等。下面的各个小节将对图 1-1 中的各个对象进行详细的说明。

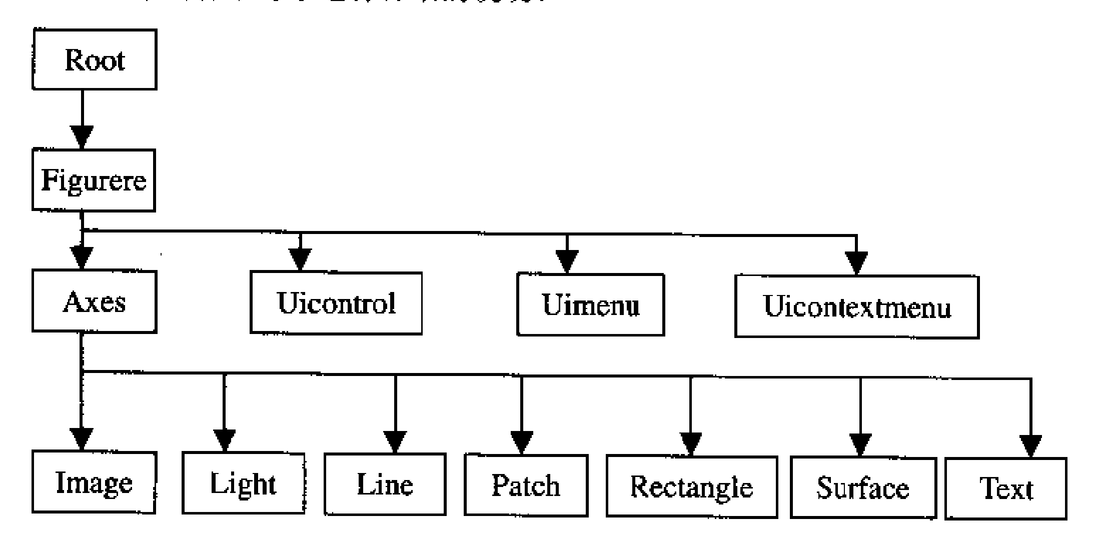


图 1-1　MATLAB 的对象从属关系图

1.2.1　Root 对象及示例

Root 对象，即根对象。它位于 MATLAB 的对象继承图的最上层，因此在 MATLAB 中创建图形对象时，只能创建惟一的一个 Root 对象，而其他的所有的对象都从属于该对象。

需要注意的是，根对象是由系统在启动 MATLAB 时自动创建的。而用户可以对根对象的属性进行设置，从而影响图形的显示效果。

1.2.2　Figure 对象及示例

Figure 对象，即图形窗口对象是作用在根屏幕（即 Root 屏幕）上显示 MATLAB 图形的独立窗口。用户可以在系统允许的前提下，在 MATLAB 中创建任意多个图形窗口。所有的图形窗口都是根对象的子对象，而所有其他的图形对象又是图形窗口对象（即 Figure 对象）的子对象。

如果当前还没有创建图形对象（即 Figure 窗口），则调用任意一个绘图函数或图像显示函数（如 plot 函数、imshow 函数等）均可以自动创建一个图形窗口。如果当前的根对象已经包含了多个图形窗口，则总有一个窗口被指定为“当前”窗口，且该窗口为所有绘



图函数的输出窗口。

在 MATLAB 中，用 figure 构造函数创建图形窗口的格式如下：

窗口句柄=figure（'属性 1'，属性值 1，'属性 2'，属性值 2，…）

figure（窗口句柄）

其中，第一种格式既产生了一个窗口句柄，同时又对窗口对象的各个属性进行了具体的设置，且按照该设置生成了一个图形窗口。

例程 1-1 创建图形窗口。

例程 1-1

```
h=figure ('color',[1 1 1],'position',[0 0 300 400],...
                'name','image processing','NumberTitle','on',...
                'menu','figure','WindowButtonDown',...
                'disp("welcome to this program")').
figure(h);
```

产生如图 1-2 所示的窗口。

另外，对于第二种格式，即 figure（窗口句柄）格式，如果窗口句柄是存在的，则调用该窗口，并使其处于可视状态，且设定为“当前”窗口；如果窗口句柄不存在，则该函数将针对这一句柄生成一个新的窗口，并设定为“当前”窗口。

例如，执行下面的这条语句：

figure(h);

将以系统默认的设置生成一个图形窗口，如图 1-3 所示。

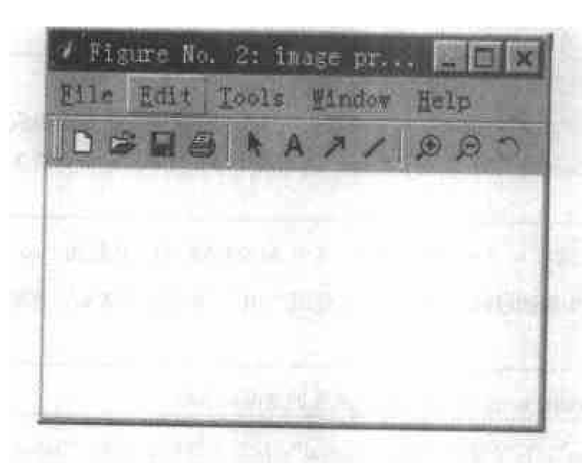


图 1-2　用户设置的图形窗口

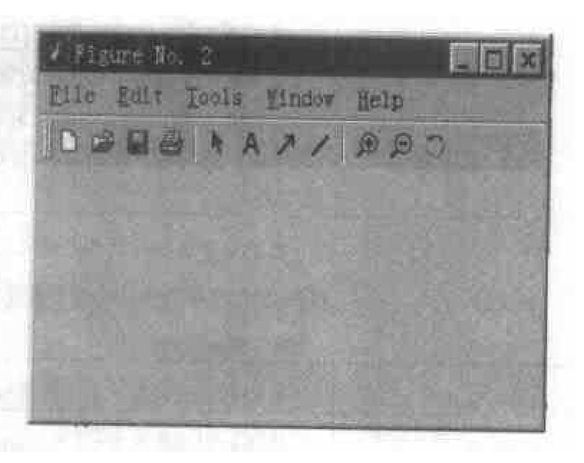


图 1-3　系统默认的图形窗口

另外，对图形窗口属性的设置还可调用 set 函数，其格式如下：

set（窗口句柄，'属性 1'，属性值 1，'属性 2'，属性值 2，…）



在调用 set 函数之前，该窗口句柄应该是已经存在的。

图形窗口的各种常用属性的设置及其可设置的属性值如表 1-1 所示。

表 1-1 图形窗口的常用属性与属性值

控 件	功 能 说 明
Color	图形的背景颜色，可设置为 3 元素的 RGB 矢量或者是 MATLAB 自定义的颜色名，MATLAB 的默认值为黑色。RGB 矢量分量的取值范围为[0 1]，取极端值可得到以下 8 种颜色：[0 0 0]黑色，[1 1 1]白色，[0 0 1]蓝色，[0 1 0]绿色，[0 1 1]天蓝色，[1 0 0]红色，[1 0 1]粉红色，[1 1 0]黄色
CurrentAxes	当前图形坐标轴的句柄
CurrentMenu	最近被选择的菜单项的句柄
CurrentObject	图形内，最近被选择的对象的句柄，可由 gco()函数获得
MenuBar	设置图形窗口菜单条形式，“figure”显示默认的 MATLAB 菜单，“none”为不加菜单。在选择“none”后，可以用 uimenu()函数设置自定义菜单，在选择“figure”后，还可以用 uimenu()函数添加新菜单
Name	设置图形窗口的标题栏内容，它的属性为一个字符串，在创建窗口时，字符串将添入标题栏
NameTitle	设置图形窗口标题栏图形标号的有无，“on”将自动地给图形窗口标题前加上 FigureNo 字样，“off”则将不在标题前加标号
Position	位置矢量[left, bottom, width, height]，[left, bottom]代表窗口左下角的坐标，Width、Height 表示窗口的宽度和高度
Units	设置尺寸的单位，“inches”：in（英寸），；“centimeters”：cm（厘米）；“normalized”：归一化坐标，取值范围[0 1]；“point”：排字点，1point =0.353mm；pixel：屏幕像素，这是 MATLAB 默认的单位
Resize	设置是否可以用鼠标调整窗口的大小，“on”可调，“off”不可调
Pointer	设置窗口下指示鼠标光标的显示形式，“crosshair”：十字形；“arrow”：箭头；“watch”：沙漏；“topl”：指向左上方的箭头；“topr”：指向右上方的箭头；“bopl”：指向左下方的箭头；“bopr”：指向右下方的箭头；“circle”：圆形光标；“cross”：双线十字；“fleur”：带有箭头的十字花形
Visible	设置窗口初始时刻是否可见，选项有“on”和“off”，其中 MATLAB 的默认值为“on”，在编程的过程当中，不必要看见中间的过程，则可以首先使用“off”，在完成设置后，再选取“on”来显示窗口
PaperSize	设置打印纸张的大小，矢量[width, height]代表了打印纸张的宽度和高度
PaperType	设置打印纸张的类型，“usletter”：标准美国信纸；“uslegal”：标准美国法定纸张；“taboid”：标准美国报纸；“a3”：A3 纸；“a5”：A5 纸；“b4”：B4 纸；“b5”：B5 纸
PaperUnits	设置纸张属性的度量单位，“inches”（英寸）：in；“centimeters”（厘米）：cm；“normalized”：归一化坐标，取值范围[0 1]；“points”：像素点，1point=0.353mm
PaperOrientations	设置打印时的纸张方向，“portrait”为纵向打印，这是 MATLAB 的默认值；“landscape”为横向打印
PaperPosition	设置打印页面上的图形位置，位置矢量[left, bottom, width, height]，[left, bottom]代表打印页面图形左下角的坐标，Width、Height 表示打印页面图形的宽度和高度



1.2.3 Uicontrol 对象及示例

Uicontrol 对象，即控制框对象，是图形用户界面（GUI）控件。当用户激活该对象时，系统将执行相应的回调函数。Uicontrol 对象的控件有多种类型，包括按钮、无线电按钮、检查框、编辑框、静态文本框、列表框、滑标等。一个良好的用户界面与这些控件是分不开的。

表 1-2 给出了控制框的控件类型及相应的属性值。

表 1-2 控制框的控件类型

控　件	说　明
按钮 (button)	按钮是对话框中最常用的控制图标，按钮上通常有提示的文本，其通常表示执行一个动作，而不是用来改变对象的状态和属性。其“style”属性值为“pushbutton”
无线电按钮 （radio）	无线电按钮是一组带有文字提示的选择项，通常用于一组互斥的选项当中，每次只能选一项。当一项被选中后，被选中按钮的圆中心显示为黑色。其“style”属性值为“radiobutton”
检查框 （checkbox）	检查框通常是由具有标志的一个小方框组成，其作用与无线电按钮极相似，主要区别是检查框可同时选择多个选项。其“style”属性值为“checkbox”
编辑框 （editbox）	编辑框是一个含有初始值的方框或无初值的空框，用户可以在里面输入自己的数据，相关的程序将会获取框中的数据信息。其“style”属性值为“checkbox”
静态文本框 （statictext）	静态文本框的主要作用是显示标志、用户提示信息和一些特定数据的当前值。其“style”属性值为“edit”
滑标 （slider）	滑标又称滚动条，它是由滚动槽、长方条区域、指示器三部分组成的。其中，长方条区域代表有效值范围，指示器代表滑标的当前值。其“style”属性值为“slider”
列表框 （listbox）	列表框是用来列出可以选择的所有选项，用户可以方便地从中选择任意选项，如果选项太多，可采用垂直滚动条加以控制。其“style”属性值为“listbox”
弹出式菜单 （popupmenu）	弹出式菜单通常用以向用户提出一系列互斥选项的列表，以使用户选择。弹出式菜单不受菜单条的限制，可以出现在窗口的任何位置，其“style”属性为“popmenu”

用户可以将 Uicontrol 对象控件组合使用，以构建控制面板和对话框。

控制框的创建用 uicontrol()函数。该函数用法如下：

控制框句柄=uicontrol（父对象的句柄，'属性 1'，属性值 1，'属性 2'，属性值 2，…）

表 1-3 给出了 Uicontrol 对象的一般属性。

表 1-3 Uicontrol 对象的一般属性

属　性	说　明
ForegroundColor	设置对象的前景颜色
BackgroundColor	设置对象的背景颜色
Callback	设置回调字符串，当控制框被选中时，执行相应的响应

（续表）

属　性	说　明
HorizontalAlignment	设置当前控件在水平方向的对齐方式，可选的值有“left”，“center”，“right”，分别代表左、中、右的对齐方式，“right”为默认值。该属性对于单个控件不起作用，只有在多个控件共存时才起作用
Max	设置属性“value”的最大有效值，且该最大值取决于 Uicontrol 对象的“type”属性。当控制框对象的无线电按钮和检查框处于“on”状态时，将 value 设定为 max；它设定了弹出式菜单最大下标值和滑标的最大下标值，当 max.min>1 时，编辑文本框为多行文本。其默认值为 1
Min	设置属性“value”的最小有效值，且该最小值取决于 Uicontrol 对象的“type”属性。当控制框对象的无线电按钮和检查框处于“off”状态时，将 value 设定为 min；它设定了弹出式菜单最小下标值和滑标的最小下标值。当 max.min>1 时，编辑文本框为多行文本。其默认值为 0
Position	设置控件的位置属性，用位置矢量[left bottom width height]来表示
String	设置控件上的字符串，在按钮、无线电按钮、检查框和弹出式菜单上指定标志
Style	设置控件类型，细节请参照表 1-2
Type	设置可读对象的辨识串，通常是用户指定的数据
Value	设置 Uicontrol 的当前值，无线电按钮和检查框处于“on”状态时，将 value 设定为 max；无线电按钮和检查框处于“off”状态时，将 value 设定为 min。滑标把 value 设置为 min≤value≤max，弹出式菜单则把 value 设置为 1≤value≤max
Visible	设置对象的可视状态，“on”为可视“off”为不可视

下面举个创建控制框对象的简单例子，更为详细的过程请参照本书的最后一部分——用户界面（GUI）的设计。

例程 1-2 创建控制框对象

例程 1-2

```
H=figure('Color',[0,1,1],'Position',[0,0,400,400],'Name',...
    '关于控制框设计的程序','NumberTitle','off', 'MenuBar','none');
uicontrol(H,'Style','text','Position',[0.05,0.74,0.9,0.1],...
        'Units','normalized','String','WELCOME  TO  THIS  PROGRAM!','Back',[0,1,1],'Fore',
[1,0,0]);
uicontrol(H,'Style','frame','Position', [0.15,0.35,0.5,0.40],'Units','normalized','Back',[1,1,0]);
uicontrol(H,'Style','edit','Position', [0.42,0.65,0.2,0.06],...
    'Units','normalized','String','256','Back',[0,1,0]);
uicontrol(H,'Style','slide','Position', [0.05,0.05,0.9,0.08],...
    'Units','normalized','Back',[1,1,0],'max',1,'min',0);
radio=uicontrol(H,'Style','radio','Position', [0.2,0.65,0.2,0.06],...
    'Units','normalized','String','输入 N','Back',[1,1,0]);
checkbox(1)=uicontrol(H,'Style','checkbox','Position', [0.2,0.54,0.2,0.06],...
```

```
    'Units','normalized','String','N=100','Back',[1,1,0]);
checkbox(2)=uicontrol(H,'Style','checkbox','Position', [0.2,0.43,0.2,0.06],...
    'Units','normalized','String','N=200','Back',[1,1,0]);
change=['disp(''改变窗口的颜色'')'];
uicontrol(H,'Style','push','Position',[0.7,0.35,0.2,0.06],...
    'Units','normalized','String','cancel','Call','close(H)');
uicontrol(H,'Style','push','Position', [0.05,0.17,0.85,0.06],...
    'Units','normalized','String','CHANGE WINDOWS ...
COLOR','Call',change);
```

执行的结果如图 1-4 所示。

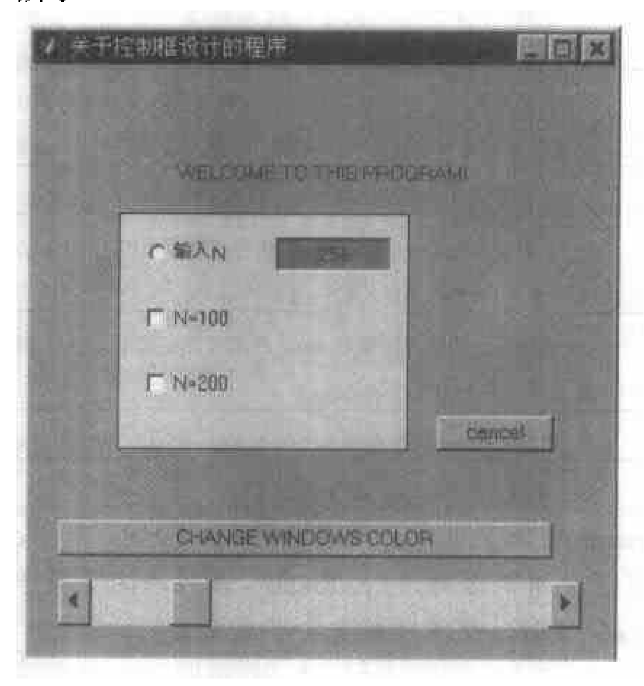


图 1-4　控制框的设计

1.2.4　Uimenu 对象及示例

在一个方便简洁的用户图形界面设计中，菜单的设计是至关重要的。有了一个完备的菜单，用户就可以方便地从菜单中选择命令和选项。MATLAB 为图形用户界面提供了一个标准的菜单，其中包括 4 个下拉菜单：【File】菜单，【Edit】菜单，【Windows】菜单和【Help】菜单。

除了上面已经提到的标准菜单外，MATLAB 还提供了 uimenu()构造函数来创建 Uimenu 对象（即下拉式菜单对象），从而设计用户感兴趣的个性化菜单。

Uimenu()函数的用法如下：

菜单项句柄=uimenu（窗口句柄，'属性 1'，属性值 1，'属性 2'，属性值 2，…）

子菜单句柄=uimenu（菜单项句柄，'属性 1'，属性值 1，'属性 2'，属性值 2，…）

菜单对象可以设置的属性如表 1-4 所示。

表 1-4　菜单对象的常用属性

属性	说明
ForegroundColor	设置菜单的前景色，可以设置为 RGB 矢量或 MATLAB 自定义的颜色名称
BackgroundColor	设置菜单的背景色，可设置为 RGB 矢量或 MATLAB 自定义的颜色名称，默认值为黑色
Callback	设置 MATLAB 的回调字符串，当该菜单被选中时，回调字符串传给函数 eval()，其初值为空矩阵
Checked	设置被选项的校验标记，如果为"on"，则校验标记出现在所选菜单的旁边；若为"off"，则不出现校验标记
Enable	设置菜单使能状态。若为"on"，则可用，当选中该菜单时执行回调函数；若为"off"，则菜单项不可用，菜单标志变灰
Label	设置菜单条名称，通常是一个字符串，在菜单项字符中可使用"&"标志以表示该符号后面的字符在显示时有一个下划线修饰，以便用户可以使用键盘【Alt】+该字符来操作菜单
Position	设置菜单项的相对排列位置，可以用来测试菜单项处于整个菜单系统的第几级
Separator	设置菜单项的分界符，"on"为在该菜单项的上面加上一条横线作为分界符，"off"为不加分界符，MATLAB 的默认值为"off"
Parent	父对象的句柄。如果 Uimenu 对象是顶层菜单，则其为图形对象；如果 Uimenu 对象是子菜单，则其为父菜单的句柄值
Visible	设置 Uimenu 对象在屏幕上的可见性，"on"为可见，"off"为不可见



Uimenu 对象属性中，最重要的是"label"和"callback"属性。"label"属性决定了菜单的名称，"callback"属性决定了菜单项的响应，如果不设置"callback"属性，菜单项将失去了意义。

例程 1-3 创建菜单对象。

例程 1-3

```
H=figure('Color',[1 1 1],'position',[0.0,300,400], 'Name','signal processing','NumberTitle','off',...
    'MenuBar','none');
mfile=uimenu(H,'label','&File');
mtran=uimenu(H,'label','&Transforms');
mhelp=uimenu(H,'label','&Help');
uimenu(mfile,'label','&New','call', 'disp(''new file selected'')');
uimenu(mfile,'label','&Open','call', 'disp(''open file selected'')');
msave=uimenu(mfile,'label','Save','Enable','on');
uimenu(msave,'label','Mfile','call','savem.m');
uimenu(msave,'label','Matfile','call','savemat.m');
uimenu(mfile,'label','Save & As','Enable','off');
uimenu(mfile,'label','&print','separator','on', 'call','print.dwin.loose.f');
uimenu(mfile,'label','&Exit','separator','on', 'call','close(H)');
uimenu(mtran,'label','&Czt','call','help czt');
```

```
uimenu(mtran,'label','&Dct','call','help dct');
uimenu(mtran,'label','&fft','call','help fft');
uimenu(mtran,'label','&Ifft','call','help ifft');
uimenu(mtran,'label','hilbert','call','help hilbert');
uimenu(mtran,'label','About...','call', 'disp("This is a menu.making example")');
```

代码执行的结果如图 1-5 所示。



(a)



(b)

图 1-5 用 uimenu 函数创建的菜单

这只是一个简单的创建菜单的例子，关于如何创建一个功能更加完善的菜单，请参照本书的最后一部分——用户界面（即 GUI）的设计。

1.2.5 Axes 对象及示例

Axes 对象，即坐标轴对象。其在图形窗口中定义一个区域，并将其所有子对象都限制在该区域内。坐标轴对象是图形窗口对象（Figure 对象）的子对象，同时又是 Image 对象（图像对象）、Light 对象（灯光对象）、Line 对象（线条对象）、Patch 对象（片块对象）、Surface 对象（表面对象）和 Text 对象（文本对象）的父对象。

如果当前运行的 MATLAB 输出窗口中还没有坐标轴对象，则任意一个绘图函数（如 plot 函数、surf 函数等）都可以创建坐标轴对象。如果在当前图形窗口中包含多个坐标轴对象，则总有一个坐标轴对象被指定为“当前”坐标轴，并且，该“当前”坐标轴即为所有绘图函数输出的坐标基准。

Axes 对象是由 axes()构造函数创建的。

axes 函数的调用格式如下：

坐标轴句柄=axes('position', rect)

其中：

rect=[left, bottom, width, height]定义了坐标系的位置和尺寸。单位归一化后，(0，0）即为当前图形窗口中左下角的点的坐标，而（1.0，1.0）则为当前图形窗口中右上角的点的坐标。

例如，执行下面的代码：

```
axes('position',[0.1,0.2,0.5,0.5]);
```

将在（0.1，0.2）处，创建一个宽度和高度均为 0.5 个单位的坐标系。结果如图 1-6 所示。

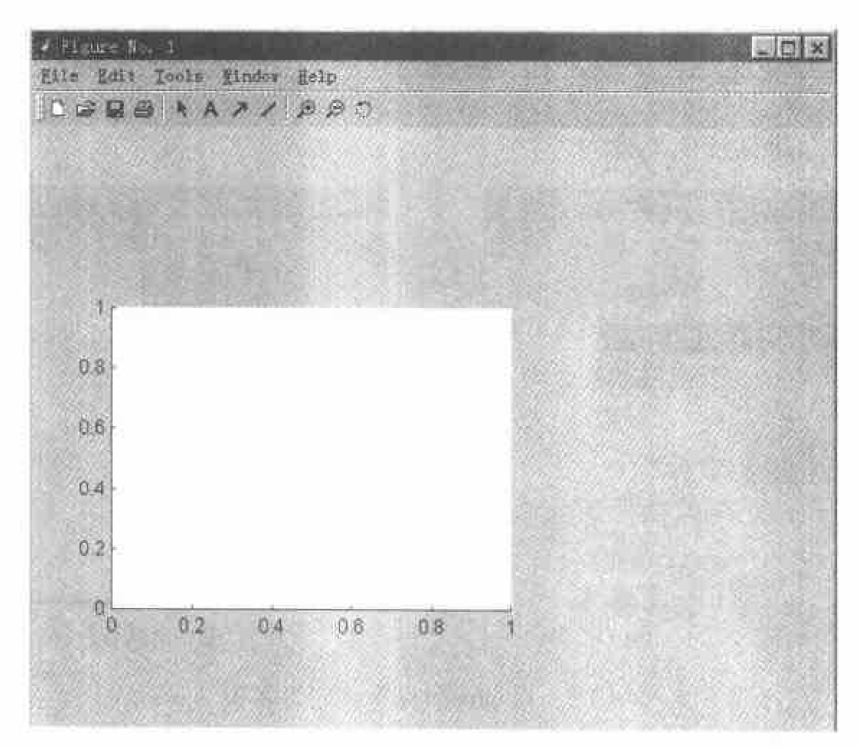


图 1-6　用 axes 构造函数创建的坐标系

另外，在 MATLAB 中，与坐标轴对象相关的函数还有如下几个：

- axis：该命令用于设置影响当前坐标轴对象的相关值。如执行命令 axis([xmin, xmax, ymin, ymax]) 可以设置当前坐标轴的坐标值范围。
- get：该命令用于获取现有的坐标轴的各项属性值。
- set：该命令用于设置现有的坐标轴的各项属性值。
- gca：该命令用于返回当前坐标轴的句柄（即标识号 ID）。

1.2.6　Image 对象及示例

Image 对象，即图像对象。它包含一个数据矩阵，有时还包含一个颜色映像表。根据数据矩阵元素代表意义的不同，MATLAB 中的 Image 对象有三种基本的类型，即索引图像、灰度图像和真彩图像。

Image 对象是由构造函数 image()来创建的。其具体用法可参见第 4 章“MATLAB 图像显示技术”。

1.2.7　Light 对象及示例

灯光设置是 MATLAB 中为增加图形场景的视觉效果而提供的一种处理技术。该项技术是通过模拟对象实体在自然光下的明暗特征来实现的。在 MATLAB 中，如果希望产生

灯光效果，必须创建一个“灯光”对象。

MATLAB 中利用 light()构造函数来创建灯光对象。每个灯光对象都包含三个非常重要的属性：

- Color：该属性用于指定来自于光源的具有方向的灯光颜色。其可以影响到图形场景中对象的显示颜色。
- Style：该属性用于指定光源类型是点光源还是无限远光源。当将 Style 设置为 local 时，表示点光源；设置为 infinite 时，表示无限远光源。默认值为后者。
- Position：该属性用于指定光线方向（对于无限远光源）或光源位置（对于点光源）。

需要注意的是，灯光对象只影响相同坐标系下的 Surface 对象和 Patch 对象。因为这两种对象才具有在有灯光对象时能够改变其自身视觉效果的相关属性。

下面这个例子，将会说明图形场景中添加灯光后的效果。

执行下面的代码：

```
set(0,'DefaultFigureColor', 'y')
membrane
```

图形窗口的颜色被设置为黄色，绘制出的表面图形如图 1-7 所示。

添加灯光的代码如下：

```
light('position',[0 .1 1],'color','r')
```

该光源为无限远光源，其中矢量[0 .1 1]定义了光源的方向，即该光源是顺序通过坐标原点(0 0 0)和(0 -1 1)；而且还定义了该光源的颜色为红色，用“r”表示。最终的效果如图 1-8 所示。

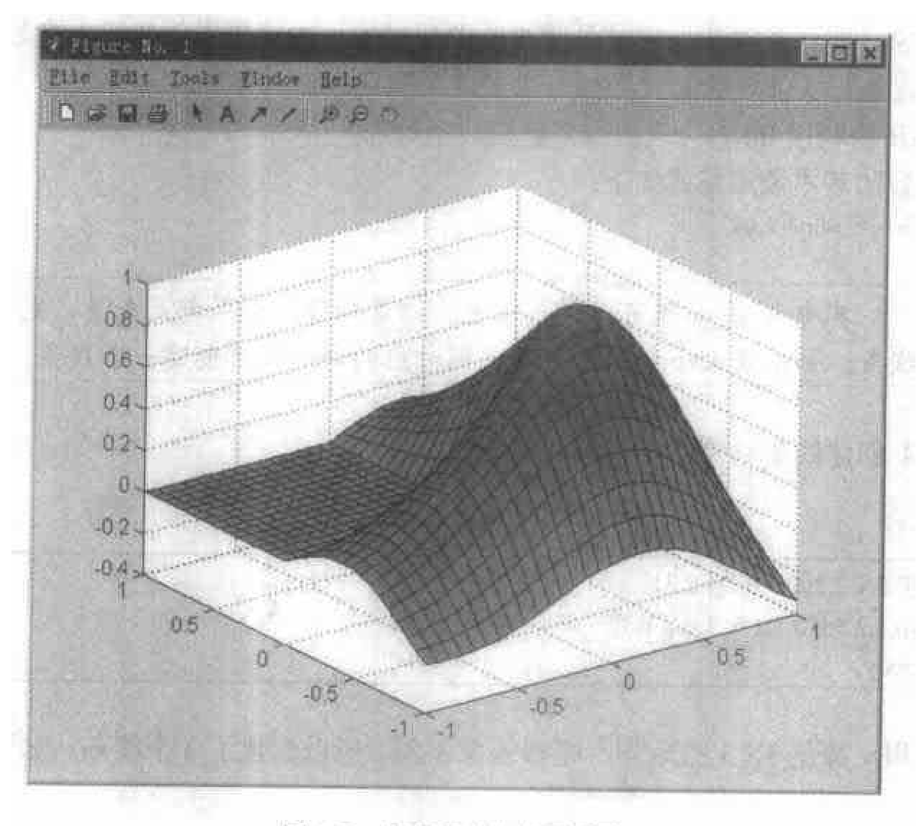


图 1-7　没有灯光的表面图形

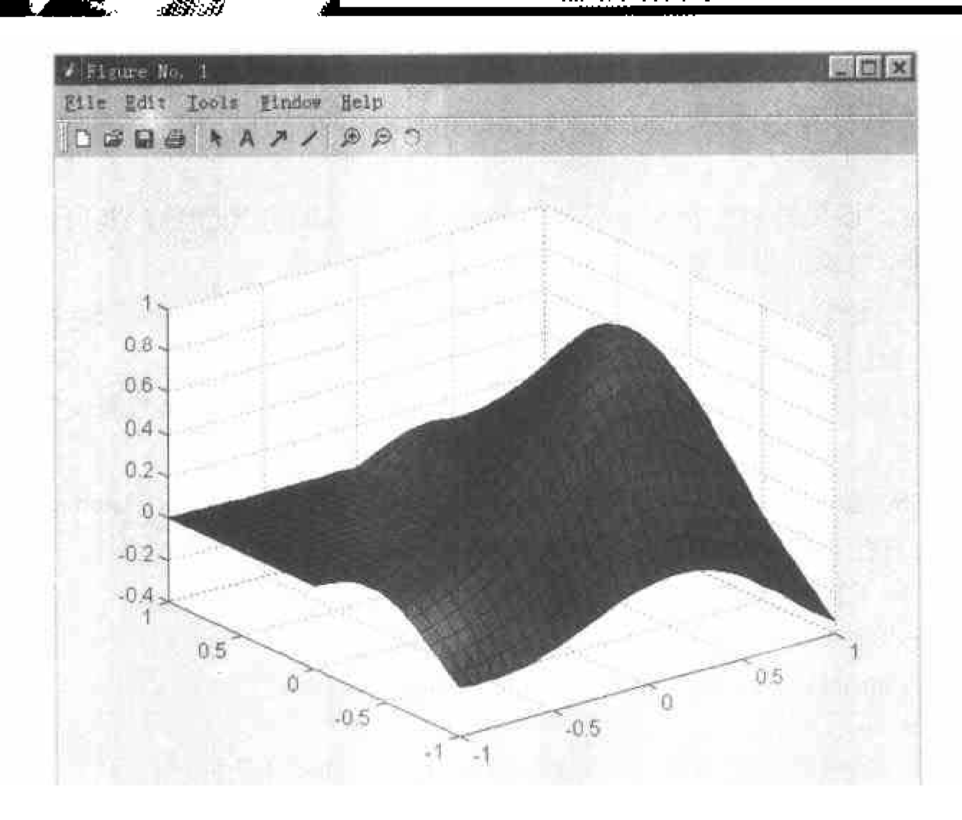


图 1-8 添加了灯光效果后的表面图形

1.2.8 Line 对象及示例

Line 对象，即线条对象。它是最基本的图形对象，主要用于创建绝大多数的二维线条和某些三维线条。线条对象的父坐标系负责线条的定位。

MATLAB 中利用 line()构造函数来创建线条对象。

line()函数的基本调用格式如下：

线条句柄=line(x,y)



其中 x 和 y 既可以是矢量也可以是矩阵。如果都是矢量，则只绘制一个线条；如果是大小相同的矩阵，则对应的每一列都创建一个线条。

例程 1-4 创建线条对象。

例程 1-4

```
x=[0.5,1.2,2;.0.5,0,1;1,2,3];
y=[0,1,2.5;0.5,1,1.5;.1,0.2,1.3];
line(x,y)
```

由于 x 和 y 都是 3×3 的矩阵，即都包含 3 列，所以创建了 3 个线条。如图 1-9 所示。

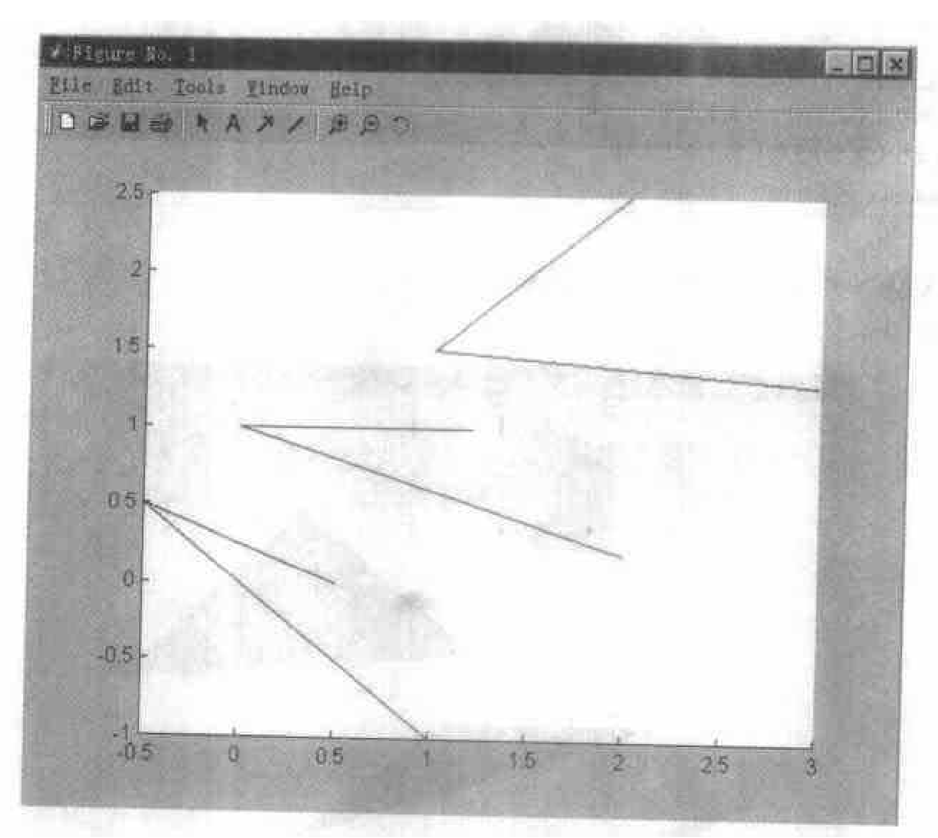


图 1-9 用 line 函数创建的线条对象



另外，MATLAB 中的许多高级函数（例如 plot 函数、plot3 函数和 loglog 函数等）也都可以创建线条对象。

1.2.9 Patch 对象及示例

Patch 对象，即片块对象。它是由一个或多个相连或不相连的多边形组成的。MATLAB 中，在对实物对象（例如飞机场、汽车等）进行建模，或绘制任意形状的二维或三维多边形时，经常会用到片块对象。

单个片块可以包含多个面，每个面均有特定的颜色。片块对象的父坐标系负责片块的定位。

MATLAB 中利用 patch()构造函数来创建片块对象。

patch()函数的基本调用格式如下：

```
patch(x,y,c)
```

其中 x 和 y 既可以是矢量也可以是矩阵。如果都是矢量，则只绘制一个片块；如果是大小相同的矩阵，则连接 x 和 y 的每一列中的各个元素所组成的点（x,y）都构成一个片块。而 c 则表示填充各个片块的颜色，其可以是字符串，如'r','g','b','c','m','y','w'和'k'，也可以是 1×3 的矢量，表示 RGB 值，如[1 0 0]代表红色。

例程 1-5 创建片块对象。

例程 1-5

```
x=[0.5,1.2,2;.0.5,0,1;1,2,3];
y=[0,1,2.5;0.5,1,1.5;.1,0.2,1.3];
patch(x,y,[1 0 0]);
```

由于 x 和 y 都是 3×3 的矩阵，即都包含 3 列，所以创建了 3 个片块，且填充颜色都是红色。如图 1-10 所示。

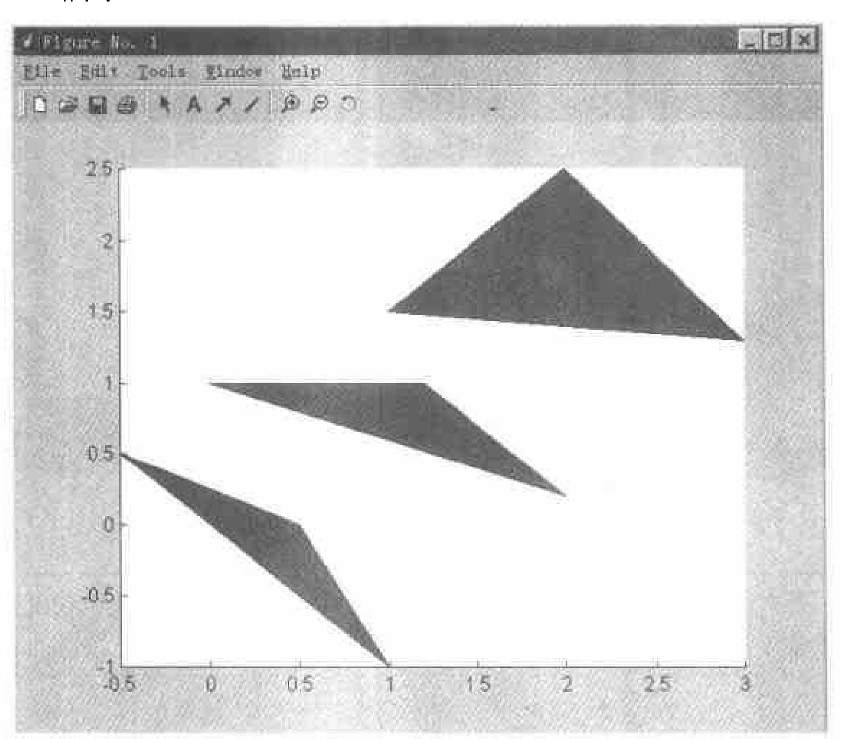


图 1-10　patch 函数创建的片块对象



另外，MATLAB 中的 fill 函数、fill3 函数和 contour 函数也都可以创建片块对象。

1.2.10 Rectangle 对象及示例

Rectangle 对象，即矩形对象。它是一个二维填充区域，其形状可以是矩形，也可以是平滑的矩形（rounded_rectangle）或椭圆。

MATLAB 中利用 rectangle()构造函数来创建矩形对象。

rectangle()函数的基本调用格式如下：

```
矩形对象的句柄=rectangle('position',[x y width height])
```

其中，(x,y)代表矩形左下角的坐标，width 代表矩形的宽度，height 代表矩形的高度。

另外，下面的代码产生的也是矩形：

```
rectangle('Curvature',[0 0])
```

下面的代码产生的是椭圆：



```
rectangle('Curvature',[1 1])
```

而下面的代码产生的则是平滑后的矩形：

```
rectangle('Curvature',[x y])
```

其中，x 和 y 取[0 1]之间的值，x 代表水平方向弯曲的程度，y 代表垂直方向弯曲的程度。

例如，下面的代码：

```
rectangle('Curvature', [0.1 0.3],'facecolor','b')
```

将创建一个平滑了的矩形，且用蓝色填充该矩形。结果如图 1-11 所示。

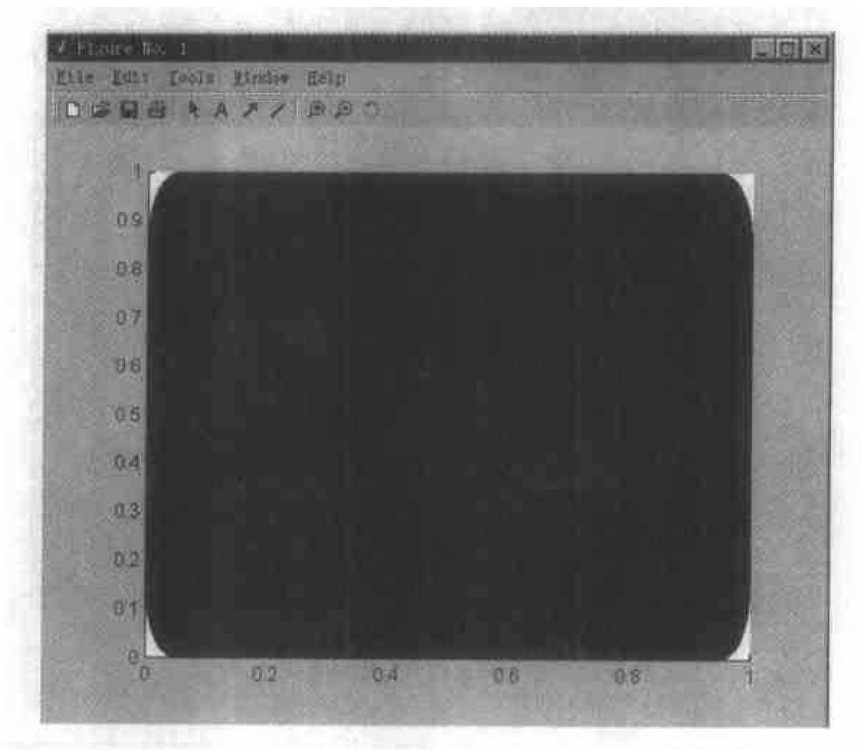


图 1-11 由 rectangle 函数创建的平滑了的矩形对象

1.2.11 Surface 对象及示例

Surface 对象，即表面对象。它是矩阵数据的三维表征。表面图形对象是由一系列的小四方形组成，四方形的每个顶点的位置是由矩阵中的对应的元素确定的。

MATLAB 中利用 surface()构造函数来创建表面对象。

surface()函数的基本调用格式如下：

```
表面图形句柄=surface(x,y,z)
```

其中 x,y,z 为三个大小相同的矩阵，三个矩阵中对应的点（x,y,z）即为该表面对象各个小四边形顶点的坐标。

例程 1-6 创建表面对象。

例程 1-6

```
x=[1 2 3;4 6 7;1 3 9];
y=[0.1 0.3 0.5;2 3 0.2;4 6 8];
```

```
z=[0.1 0.3 0.5;2 3 0.2;4 6 8];
h=plot3(x,y,z);
delete(h);
surface(x,y,z);
```

创建了一个表面对象，结果如图 1-12 所示。

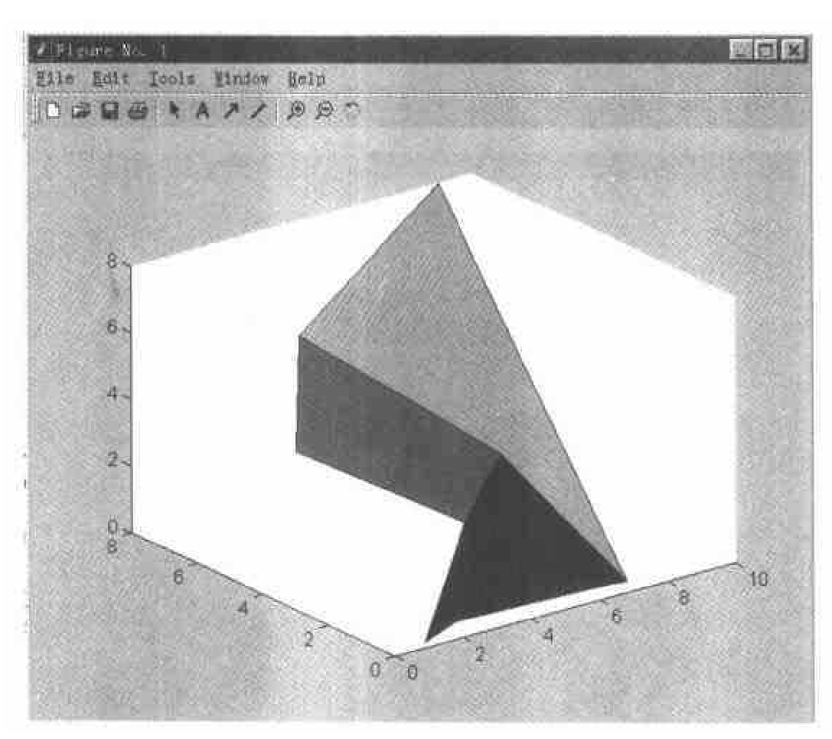
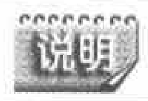


图 1-12 由 surface 函数创建的表面对象



MATLAB 中的高级函数 surf 和 pcolor，以及网格绘制函数均可以创建表面对象。

1.3 图形对象的属性

图形对象的属性控制图形的外观和显示特点。图形对象的属性包含公共属性和特有属性。

MATLAB 将图形信息组织在一个有序的金字塔式的阶梯图表中，并将其存储在对象属性中。例如，根对象的属性包含当前图形窗口对象（Figure 对象）的句柄和鼠标指针的当前位置；而图形窗口对象属性则包含其子对象的列表，同时跟踪发生在当前图形窗口中的某些 Windows 事件；坐标轴对象属性则包含其每个子对象（图形对象）使用图形颜色映射表的信息和每个绘图函数对颜色的分配信息。

通常情况下，用户可以随时查询和修改绝大多数属性的当前值，而有一部分属性对用户来说是只读的，只能由 MATLAB 修改。需要注意的是，任何属性只对某个对象的某个具体的实例才有意义，所以，修改同一种对象的不同实例的相同属性时，彼此互不干涉。

用户可以为对象属性设置默认值，则此后创建的所有该对象的实例所对应的这个属性的值均为该默认值。

表 1-5 给出了 MATLAB 中图形对象的公共属性。

表 1-5 MATLAB 图形对象的公共属性

属 性	说 明
BusyAction	控制 MATLAB 图形对象句柄回调函数的中断方式
ButtonDownFcn	当单击按钮时执行回调函数
Children	该对象所有子对象的句柄
Clipping	使能或非使能剪切模式（只对坐标轴子对象有效）
CreateFcn	当同种类型的对象创建时执行回调函数
DeleteFcn	当用户发出删除该对象的指令时执行回调函数
HandleVisibility	容许用户控制来自于命令行和回调函数内部的对象句柄是否可用
Interruptible	确定当前的回调函数是否可以被后继的回调函数中断
Parent	该对象的父对象
Selected	表明该对象是否被选中
SelectionHighlight	指定选中的对象是否以可视化的方式显示出来
Tag	用户指定的对象标签
Type	该对象的类型（是 Figure 对象、Line 对象，还是 Text 对象等）
UserData	用户想与该对象关联的任意数据
Visible	指明该对象是否可见

1.4 图形对象属性值的设置和查询

在创建 MATLAB 的图形对象时，通过向构造函数传递“属性名/属性值”参数对，用户可以为对象的任何属性（只读属性除外）设置特定的值。构造函数返回其创建的对象句柄，利用该句柄，用户可以在对象创建完成后对其属性值进行查询和修改。

在 MATLAB 中，set 函数用于设置现有图形对象的属性值，get 函数用于返回现有图形对象的属性值。利用这两个函数，还可以列出具有固定设置的属性的所有值。

下面具体介绍如何利用这两个函数来实现对图形对象属性值的设置和查询。

1.4.1 属性值的设置

MATLAB 中，用户利用 set 函数和对象相应的构造函数的返回句柄，可以设置该对象的各项属性。

set 函数的基本调用格式如下：

```
set(对象句柄, '属性 1', 属性值 1, '属性 2', 属性值 2, …)
```

例程 1-7 表示 set 函数基本调用。

例程 1-7

```
t=0:pi/20:2*pi;
z=sin(t);
plot(t,z);
set(gca,'YAxisLocation','right);     %设置当前坐标轴的 Y 轴的位置属性为右
xlabel('x');
ylabel('y');
```

执行结果将 *Y* 轴移到了当前坐标系的右边。结果如图 1-13 所示。

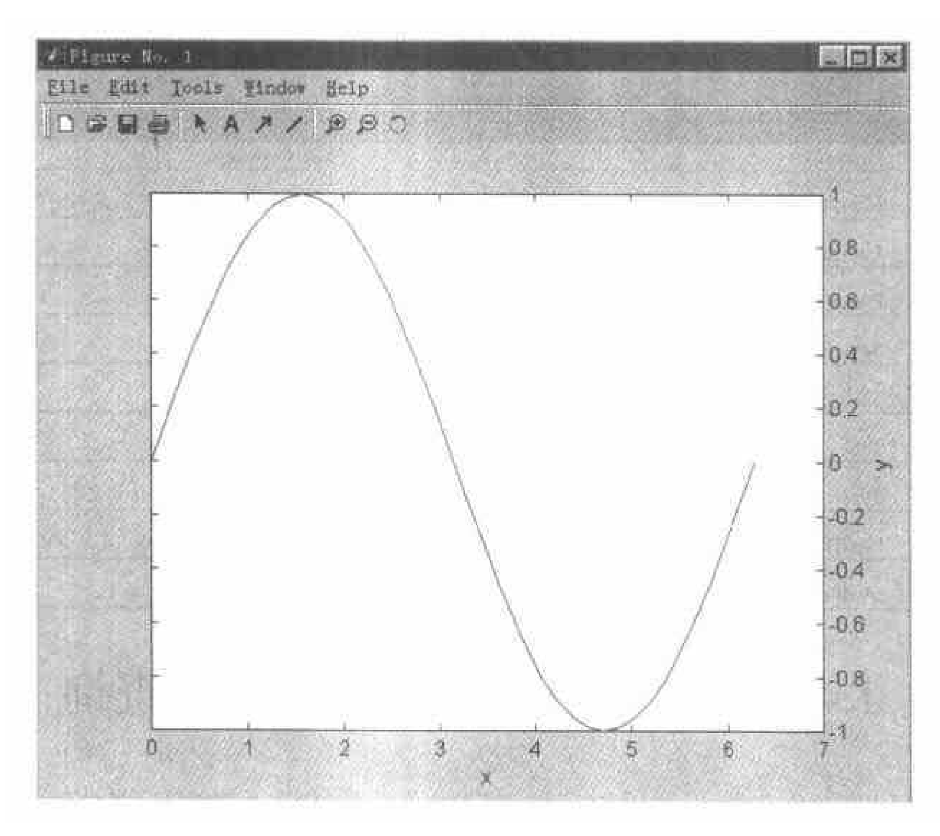


图 1-13　将 *Y* 轴移到坐标系右边的图形显示

如果对象句柄是一个矢量，则 set 函数对该矢量中所有的对象均可以进行属性的设置。

另外，利用 set 函数可以得到多个属性的所有可能的值。例如，执行下面的代码将会得到用户设定的 Line 对象（即线条对象）的线形属性的所有值：

```
set(line,'LineStyle')
```

得到 Line 对象的属性 LineStyle 的属性值列表如下：

```
[ {.} | .. | · | .. | none ]
```

其中，大括号内表示默认值。

如果想查看一个对象，如 Line 对象的所有可设置属性（即非只读属性）的所有可能的值，则使用如下的格式：

```
set(line)
```

得到如下的结果：

```
Color
```

```
    EraseMode: [ {normal} | background | xor | none ]
    LineStyle: [ {.} | .. | : | .. | none ]
    LineWidth
Marker: [ + | o | * | . | x | square | diamond | v | ^ | > | < | pentagram | hexagram | {none} ]
    MarkerSize
    MarkerEdgeColor: [ none | {auto} ] .or. a ColorSpec.
    MarkerFaceColor: [ {none} | auto ] .or. a ColorSpec.
    XData
    YData
    Zdata
  .
  .
  .
UserData
    Visible: [ {on} | off ]
```

1.4.2　对象的默认属性值

在 MATLAB 中，所有的对象属性均有系统默认的属性值，即工厂（factory）设定值。同时，用户也可以自己定义任何一个 MATLAB 对象的默认属性值。

1．默认属性值的搜索

MATLAB 对默认属性值的搜索是从当前对象开始的，沿着对象的从属关系图向更高的层次搜索，直到发现系统的默认值或用户自己定义的值。

定义对象的默认值时，在对象从属关系图中，该对象越靠近 Root 根对象，其作用的范围就越广。例如，如果我们在根对象的层次上为 Line 对象定义了一个默认值，由于根对象位于对象从属关系图的最上层，所以该值将会作用于所有的线条对象。

如果用户在对象从属关系图的不同层次上定义同一个属性的默认值，则 MATLAB 将会自动选择最下层的属性值作为最终的属性值。需要注意的是，用户自定义的属性值只能影响到该属性值设置后创建的对象，之前和现有的对象都不受影响。

2．默认属性值的设置

指定 MATLAB 对象的默认值，要首先创建一个以“Default”开头的字符串，该字符串的中间部分为对象类型，最末尾为对象的属性名称。

例如，指定当前图形窗口对象层上线条对象的线宽属性的默认值的代码如下：

```
set(gcf,'DefaultLineLineWidth',2.5)
```

其中，字符串 DefaultLineLineWidth 标识了所要设置默认值的属性为线条对象的线宽属性。而句柄 gcf 则指示了所设置属性值所在的对象层，此处即为当前图形窗口对象层。

执行完上述代码后，再执行下面的代码：

```
line
```

得到如图 1-14 所示的结果。

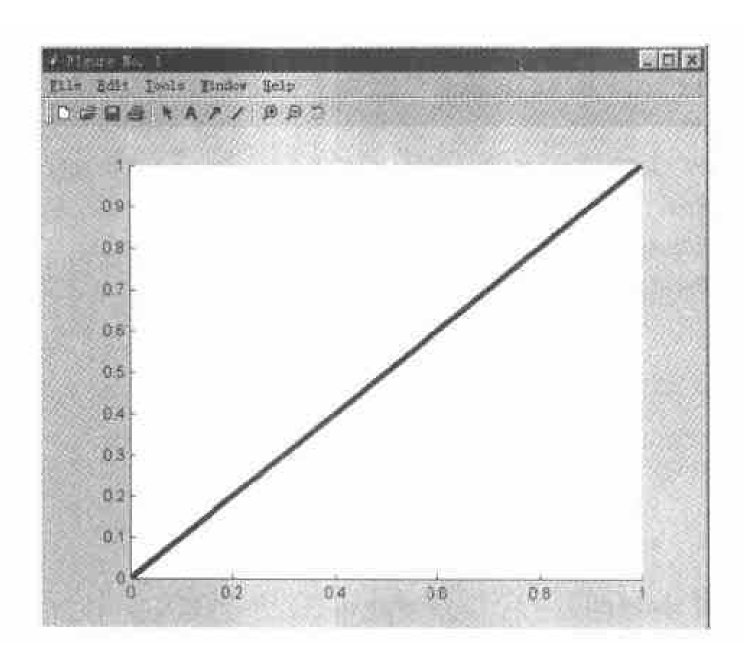


图 1-14 将线宽设置为 2.5 后所绘制的直线

又如下面的代码：

```
set(0,'DefaultFigureColor', 'w')
```



使用字符串 DefaultFigureColor 指定图形窗口的默认颜色。此处的句柄参数为 0，表示是在根对象层指定图形窗口的默认颜色。

另外，利用 get 函数可以得到当前某一对象层上所有默认的设置。例如执行下面的代码：

```
set(gcf,'DefaultLineLineWidth',2.5)
get(gcf,'default')
```

得到：

```
ans =
lineLineWidth: 2.5000
```

即表示当前图形窗口中所有的默认设置只有一项，即 lineLineWidth，且值为 2.5000。

其实，将一个属性值设置为默认（即 default），也就是将该值设置为该属性所有可能值中的第一个值。

1.4.3 属性值的查询

在 MATLAB 中，利用 get 函数可以查询对象属性的当前值。

1. 对于某个指定的属性值的查询

对于对象的某个指定的属性值的查询，其格式如下：

```
get(对象句柄, '属性')
```

例如，我们要查询当前的图形窗口对象的颜色映像表的属性，其代码如下：

```
get(gcf,'colormap')
```

执行后，得到如下的结果：

```
ans =
0         0      0.5625
0         0      0.6250
0         0      0.6875
0         0      0.7500
0         0      0.8125
.
.
.
0.8125    0      0
0.7500    0      0
0.6875    0      0
0.6250    0      0
0.5625    0      0
```

2. 对象所有属性值的查询

对于对象的所有属性值的查询，其格式如下：

```
get(对象句柄)
```

例如，我们要查询当前的坐标轴对象的所有的属性，其代码如下：

```
get(gca)
```

执行后，得到如下的结果：

```
AmbientLightColor = [1 1 1]
Box = off
CameraPosition = [-40.6571 -43.6012 38.641]
CameraPositionMode = auto
CameraTarget = [5 4 4]
.
.
.
UserData = []
Visible = on
```

另外，我们也可以把由 get 函数得到的对象的多项属性值赋给一个变量，而且 MATLAB 自动将其创建成了一个结构体，而对象的各项属性即构成了结构体中的各个元素。如果我们要访问对象的某一项属性，只需引用“结构体变量.属性名”即可。需要注意的是，原本属性名是不区分大小写的，但是该处属性名要区分大小写，否则会提示出错。

例如，执行下面的代码：

```
h=plot([1 2 3],[2 5 7]);
a=get(h);
```

表示我们将由 plot 函数创建的线条对象的属性值赋给了变量 a。再执行下面的代码：

```
a.Color
```

得到该线条对象的颜色属性值为：

```
ans =
0      0      1
```

如果执行下面的代码：

```
a.color
```

则会提示下面的错误信息：

```
??? Reference to non.existent field 'color'.
```

3．属性的系统设定值（也称为工厂设定值）的查询

由于 MATLAB 为所有的图形对象的所有属性都指定了一个特定的值，所以当用户没有把属性值作为参数使用或没有指定其默认值时，所有的属性值都取系统指定的这个特定的值。

执行下面的代码：

```
a=get(0,'factory')
```

我们将得到所有属性的系统设定值，而且这些属性值都作为其中的一个元素赋给了结构体变量 a。

另外，利用下面的代码，我们也可以获得单个属性的系统设定值：

```
get(0,'factory 对象类型属性名')
```

例如，执行下面的代码，我们将得到坐标轴对象的颜色属性的系统设定值：

```
get(0,'factoryaxescolor')
```

需要注意的是，调用函数 get 来得到属性的系统设定值的过程中，对象句柄只能为 0，即只能是根对象。

1.5　图形对象句柄的访问

MATLAB 为其所创建的所有的图形对象都赋予了一个句柄。所有的构造函数和一些高级函数的返回值即为其所创建对象的句柄。如果用户需要访问某一对象的属性（例如在 M 文件中），就应该在创建对象的时候就将其句柄保存在某一用户指定的变量中，以备后来调用。

当然，MATLAB 中也提供了专门的函数即 findobj 函数来获取现有对象的句柄。另外，罗列其父对象的 Children 属性也可以获取现有对象的句柄。

1.5.1　图形对象句柄的取值

每个图形对象句柄都有一个特定的值。但不同的对象，句柄的取值类型也略有不同。

Root 根对象的句柄取值恒为 0。

Figure 对象，即图形窗口对象，其 IntergerHandle 属性控制该对象所获取的句柄的取值类型。图形窗口对象的句柄取值通常有两种情况：

- 整数。此为默认的情况，显示在窗口标题栏中。如'Figure No. 1'中的 1 即为当前图形窗口对象的句柄的取值。此时 IntergerHandle 属性为“on”。
- 具有完全 MATLAB 内部精度的浮点数。此时 IntergerHandle 属性为“off”。

此外，所有其他的图形对象的句柄取值均为浮点类型。在引用这些句柄时，一定要维护其精度的完整。



1.5.2 句柄图形的当前形

在MATLAB中，句柄图形有一个非常重要的概念就是当前形（即Being Current）。例如，当前的图形窗口即为接受绘图函数输出的窗口，同样，当前坐标轴就是创建坐标子对象的命令的输出目标，而当前的图形对象即为最后创建的图形对象，或最后被鼠标点中的图形对象。

通常情况下，MATLAB保存三个"当前句柄"。其关系如图1-15所示。

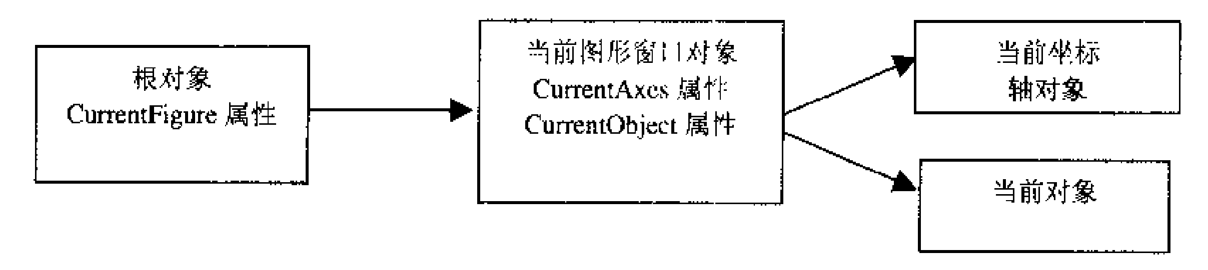


图1-15 MATLAB的句柄结构

这些属性能够使得用户方便地获取这些关键对象的句柄，其方法如下：

- get(0,'CurrentFigure')：获取当前图形窗口对象的句柄。
- get(gcf,'CurrentAxes')：获取当前图形窗口对象的当前坐标轴对象的句柄。
- get(gcf,'CurrentObject')：获取当前图形窗口对象中当前对象的句柄。

对于上述三种格式，MATLAB提供了三个简化的格式，分别对应如下：

- gcf
- gca
- gco

我们可以利用上面的这些命令作为句柄参数传给相关的函数。例如，我们可以首先单击一个线条对象，然后调用gco命令，将其返回的句柄作为参数传递给set函数，代码如下：

```
set(gco,'Marker','square')
```

而列出当前坐标轴对象的所有属性值的代码如下：

```
set(gca)
```

此外，用户可以获取当前坐标系下所有图形对象的句柄，代码如下：

```
h=get(gca,'Children');
```

然后，据此确定对象类型，代码如下：

```
get(h,'type')
```

1.5.3 通过属性值查找对象

MATLAB的findobj函数提供了一种方法用于快速遍历对象从属关系表，并获取具有特定属性值的对象句柄。如果用户没有指定起始对象，那么findobj函数就从根对象开始查找。

findobj 函数的调用格式如下：

```
h = findobj
h = findobj (ObjectHandles)
h = findobj ('P1Name',P1Value,…)
h = findobj (ObjectHandles, 'P1Name', P1Value,…)
h= findobj (ObjectHandles, 'flat', 'P1Name', P1Value,…)
```

说明：

h = findobj 表示返回根对象和它的所有子对象的句柄，返回值为一个列矢量。

h = findobj (ObjectHandles)表示返回由句柄 ObjectHandles 所确定的对象表中的对象以及它们的所有子对象的句柄。

h = findobj ('P1Name',P1Value,…)表示从根对象开始，遍历对象从属关系表，从而获取具有特定属性值的对象句柄。

h = findobj (ObjectHandles, 'P1Name', P1Value,…)表示搜索的范围只能是由句柄 ObjectHandles 所确定的对象表中的对象及它们所有的子对象。

h= findobj (ObjectHandles, 'flat', 'P1Name', P1Value,…) 表示搜索的范围只能是由句柄 ObjectHandles 所确定的对象表中的对象，而不需要搜索它们的子对象。

例程 1-8 创建一个带有标注的正弦曲线。

例程 1-8

```
t=0:pi/20:2*pi;
plot(t,sin(t));
xlabel('t=0 to 2pi');
ylabel('sin(t)');
title('0 到 2pi 的正弦曲线');
text(pi/4,sin(pi/4),'\leftarrowsin(t)=0.707');
```

显示的结果如图 1-16 所示。

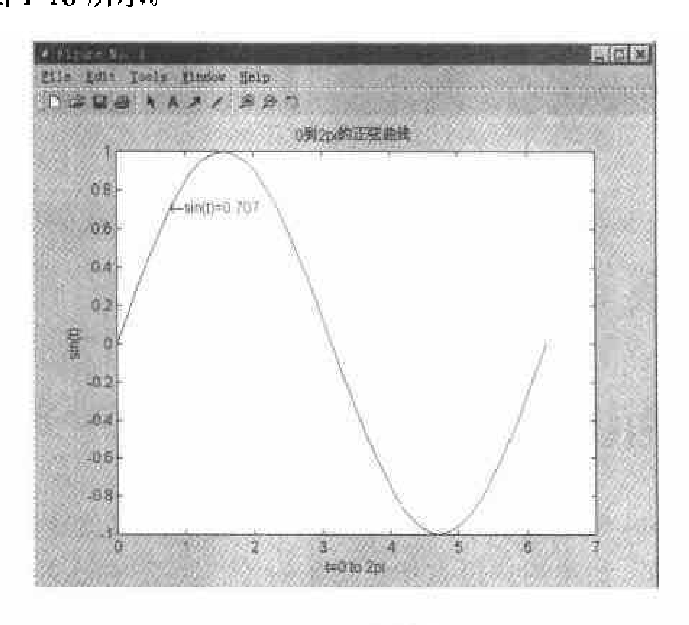


图 1-16 正弦曲线图



下面我们通过 findobj 函数来获取文本对象的句柄，进而实现对文本字符串的移动。代码如下：

```
text_handle=findobj('string','\leftarrowsin(t)=0.707');
set(text_handle,'position',[3/4*pi,sin(3/4*pi),0]);
```

文本移动后的结果如图 1-17 所示。

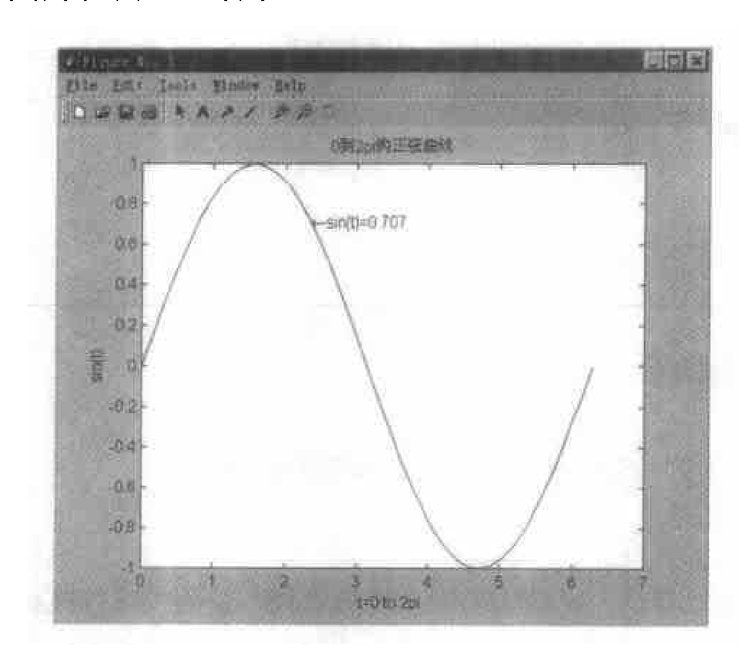


图 1-17　文本移动后的正弦曲线图

1.5.4　图形对象的拷贝

利用 MATLAB 中的 copyobj 函数，可以将一个图形对象从其父对象中拷贝到另一个对象中，从而成为该对象的子对象。当然，要实现这种拷贝，拷贝对象与新的父对象之间必须是一种正确的父对象/子对象关系，即要符合对象从属关系。

copyobj 函数的调用格式如下：

```
obj_handle=(h,p)
```

需要说明的是，h 和 p 的取值有三种情况：

1）当 h 和 p 都是矢量时，两者的维数必须相同，所返回的对象句柄也是一个与它们的维数相同的矢量。且 obj_handle(i)代表将 h(i)拷贝到新的父对象 p(i)后得到的新的句柄。

2）当 h 是矢量，而 p 是标量时，所返回的对象句柄是与 h 维数相同的矢量。且 obj_handle(i)代表将 h(i)拷贝到新的父对象 p 后得到的新的句柄。

3）当 h 是标量，而 p 是矢量时，所返回的对象句柄是与 p 维数相同的矢量。且 obj_handle(i)代表将 h 拷贝到新的父对象 p(i)后得到的新的句柄。

由以上的说明可以看出，拷贝后的对象与原来的对象之间存在着两个不同：

- 父对象不同
- 对象句柄不同


当拷贝具有子对象的对象时，该对象所有的子对象将同时被拷贝。

例程 1-9 把一个正弦曲线拷贝到一个指定的坐标轴对象中。

例程 1-9

```
figure;
t=0:pi/20:2*pi;
plot(t,sin(t));

f=axes;
copyobj(h,f);
```

1.5.5 图形对象的删除

利用 MATLAB 中的 delete 函数，可以删除图形对象。

delete 函数的调用格式如下：

```
delete(obj_handle)
```

由上面的格式可以看出，要想删除一个图形对象，必须先获得该对象的句柄。


当删除一个具有子对象的对象时，该对象所有的子对象将同时被删除。

例程 1-10 把当前的坐标轴连同它的子对象——一条正弦曲线同时从图形窗口中删除。

例程 1-10

```
t=0:pi/20:2*pi;
plot(t,sin(t));
delete(gca);
```

第 2 章　MATLAB 图形绘制

MATLAB 为用户提供了大量的绘图函数，使用户可以方便地实现数据的可视化，如在直角坐标系或极坐标系中绘制直线、条形图、柱状图、轮廓线和表面网格图等。本章将对这些绘图函数的用法进行较为详细的介绍，使读者能够对 MATLAB 提供的这些高级绘图函数有较为深刻的认识，从而绘制出自己需要的图形。

本章主要内容：

- 基本曲线的绘制
- 条形图的绘制
- 柱状图的绘制
- 饼图的绘制
- 离散数据的图形绘制
- 方向和速度矢量图的绘制
- 轮廓图的绘制
- 动画的绘制

2.1　基本曲线的绘制

MATLAB 提供了大量的图形绘制函数将矢量数据以曲线图形的方式进行显示，而且用户还可以通过设置对象的属性（如线条类型、线条的颜色等）来控制显示的外观。

表 2-1 列出了 MATLAB 中绘制基本曲线图形的函数。

表 2-1　基本的绘制曲线的函数

绘图函数	用途
plot	在线性坐标系中绘制二维数据
plot3	绘制三维数据
loglog	在对数坐标系中绘制图形
semilogx	在半对数坐标系中绘制图形（X 轴为对数比例）
semilogy	在半对数坐标系中绘制图形（Y 轴为对数比例）
plotyy	在双 Y 轴坐标系中绘制图形

下面就简单地介绍一下这些函数的用法。

- plot 函数

plot 函数的调用格式如下：

```
plot(Y)
plot(X1,Y1,…)
```

```
plot(X1,Y1,LineSpec, …)
plot(…,'PropertyName',PropertyValue, …)
h = plot(…)
```

其中，plot(Y)中如 Y 为一个矢量，则绘制连接(i ,Y(i))的折线；若 Y 中的 1 个元素为复数，则绘制连接（real(Y(i)),imag(Y(i))的折线；若 Y 为矩阵，则将每一列都视为一个矢量绘制一条折线。

在 plot(x,y)中，若 x 和 y 都是矢量，则绘制出连接(x(i)，y(i))的折线。x 和 y 也可以是矩阵，但要注意维数的匹配。

参数 LineSpec 表示定义线形、标记符号、颜色，以后凡遇到该参数，若无特别说明，都是此含义。

- plot3 函数

plot3 函数的调用格式如下：

```
plot3(X1,Y1,Z1,…)
plot3(X1,Y1,Z1,LineSpec, …)
plot3(…,'PropertyName',PropertyValue, …)
h = plot3(…)
```

其中 *X1*，*Y1* 和 *Z1* 若是三个长度相同的矢量，则绘制连接(X(i),Y(i),Z(i))的折线；若为大小相同的矩阵，则每一列都绘制一条折线。

- loglog 函数、semilogx 函数和 semilogy 函数

这三个函数的调用格式与 plot 函数的格式相同，只不过是显示的坐标轴的比例不同，此处就不多介绍了。

- plotyy 函数

plotyy 函数的调用格式如下：

```
plotyy(X1,Y1,X2,Y2)
plotyy(X1,Y1,X2,Y2,'function')
plotyy(X1,Y1,X2,Y2,'function1','function2')
[AX,H1,H2] = plotyy(…)
```

参数'function'代表绘制数据的方式，可选择'plot' ,'semilogx','semilogy','loglog','stem'等。

[AX,H1,H2] = plotyy(…)中的 AX 是返回的两个坐标轴的句柄，H1 和 H2 为两个图形对象的句柄。

例程 2-1 绘制一个带标注的双 *Y* 轴图形。

例程 2-1

```
t=0:1000,
A=1000;
a=2.005;
b=2.005;
z1=A*exp(.a*t);
z2=cos(b*t);
[haxes,hline1,hline2]=plotyy(t,z1,t,z2,'semilogy','plot');
%获取两个坐标轴和两个线条对象的句柄
axes(haxes(1));
```

```
ylabel('semilog plot');        %标注左边的 Y 轴
axes(haxes(2));
ylabel('linear plot');         %标注右边的 Y 轴
```

显示的结果如图 2-1 所示。

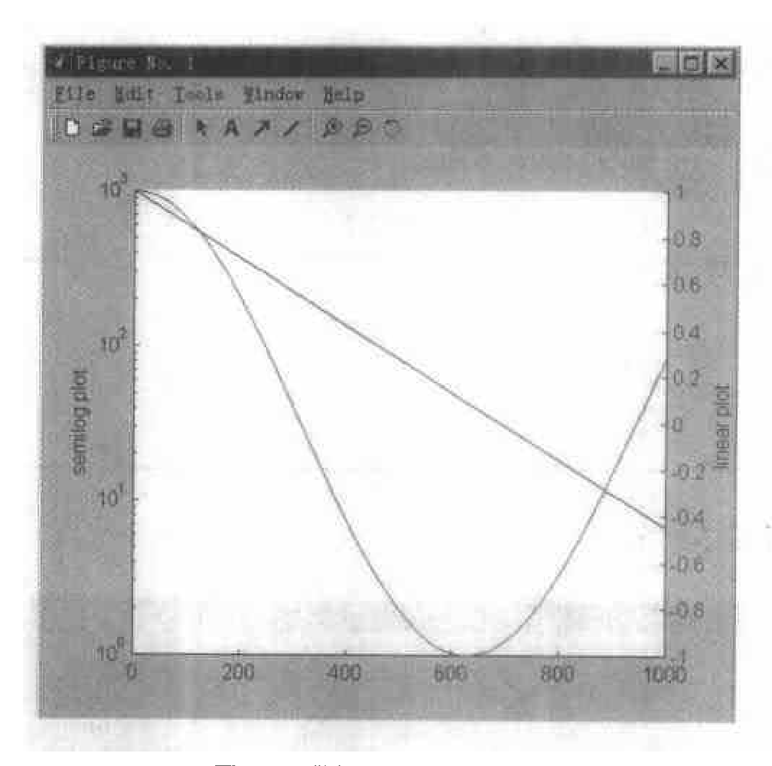


图 2-1 带标注的双 Y 轴图形

2.2 条形图的绘制

MATLAB 中有四类条形图：

- 二维垂直条形图
- 三维垂直条形图
- 二维水平条形图
- 三维水平条形图

2.2.1 二维垂直的条形图

在 MATLAB 中，用函数 bar 来绘制二维条形图。该函数的调用格式如下：

```
bar(Y)
bar(x,Y)
bar(…,width,'style',LineSpec)
```

bar(Y)中的 Y 为 $m \times n$ 的矩阵，绘制后的条形图中有 m 组，每组有 n 个条形，且第 i 组在 X 轴的中心坐标为 i，条形的高度为所对应的元素的大小。

bar(x,Y)与 bar(Y)的意义类似，但第 i 组在 X 轴的中心坐标为 x(i)。另外，需要注意的是，此处 X 中的各元素必须是降幂或升幂排列。

bar(…,width,'style',LineSpec)中的参数 width 代表条形的宽度，当 width 的值大于 1 时，条形将会出现交叠，默认值为 2.8。参数'style'用来定义条形的类型，可选值为“group”或“stack”，默认值为“group”，当选“stack”时，对于 $m\times n$ 的矩阵只绘制 n 组条形，每组一个条形，且该条形的高度为这一列中所有元素的和。参数 LineSpec 用来定义条形的颜色。

例程 2-2 绘制一个常规的二维条形图。

例程 2-2

```
x=[2.5 1.5 2];
y=[2 4 1;
   3 6 2;
   2 4 8];
bar(x,y,1.2);
```

显示的结果如图 2-2 所示。

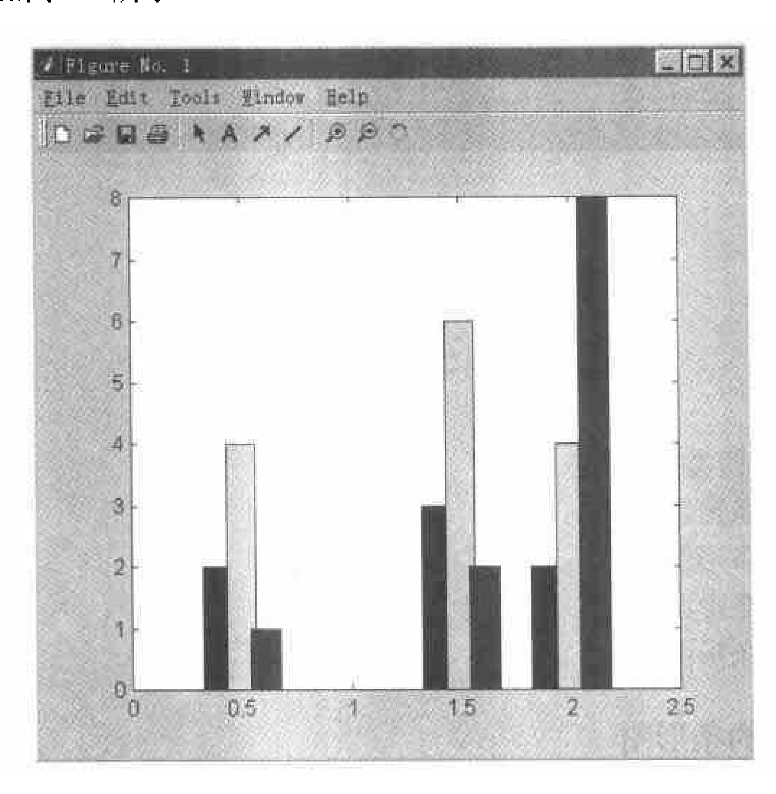


图 2-2 常规的二维条形图

例程 2-3 绘制一个堆叠的二维条形图。

例程 2-3

```
x=[2.5 1.5 2];
y=[2 4 1;
```

```
    3 6 2;
    2 4 8];
bar(x,y,1.2,'stack');
%此时将参数'style'设置为'stack'
```

显示的结果如图 2-3 所示。

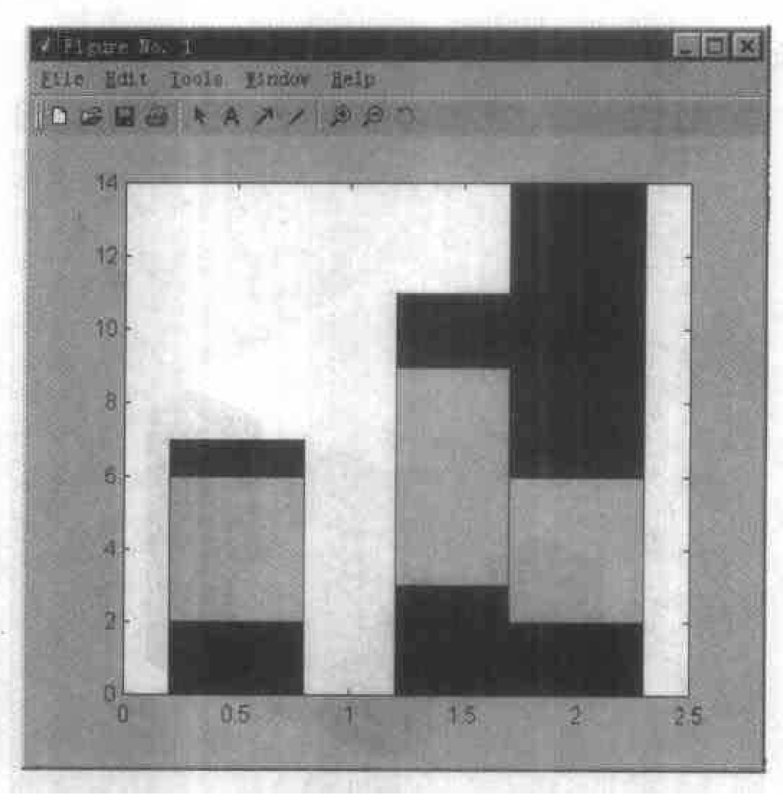


图 2-3 堆叠的条形图

2.2.2 三维垂直的条形图

在 MATLAB 中，用函数 bar3 来绘制三维条形图。该函数的调用格式如下：

```
bar3(Y)
bar3(x,Y)
bar3(…,width,'style',LineSpec)
```

与二维条形图不同的是，'style'还可以取“detached”，此时在 X 轴方向各个实心块是彼此分离的。

另外，需要说明的是，三维条形图的各组的实心块是沿着 Y 轴分布的，而不同的组是沿着 X 轴排列的。

例程 2-4 创建一个分离的条形图。

例程 2-4

```
x=[2.5 1.5 2];
y=[2 4 1;
   3 6 2;
```

```
    2 4 8];
bar3(x,y,'detached');
xlabel 'x 轴';
ylabel 'y 轴';
zlabel 'z 轴';
set(gca,'xtick',[1 2 3]);      %设置 X 轴的标度
```

显示的结果如图 2-4 所示。

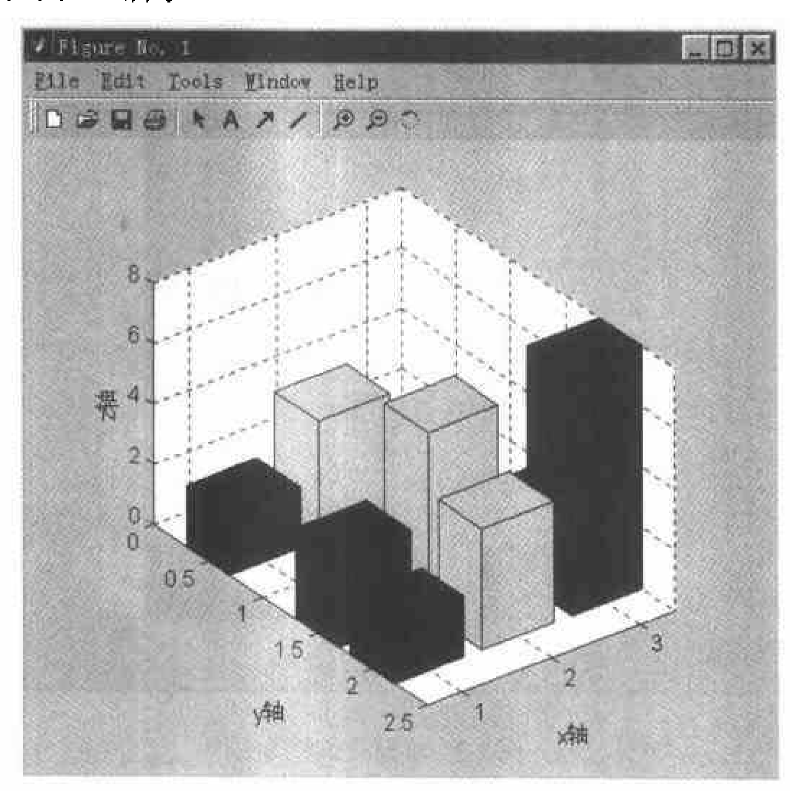


图 2-4　分离的三维条形图

2.2.3　二维水平的条形图

MATLAB 中，用 barh 函数来绘制二维水平条形图。其格式和用法与函数 bar 相同。只是绘出的条形图是水平的。

例程 2-5 绘制一个二维水平日堆叠的条形图。

例程 2-5

```
x=[2.5 1.5 2];
y=[2 4 1;
    3 6 2;
    2 4 8];
barh(x,y,'stacked');
```

显示的结果如图 2-5 所示。

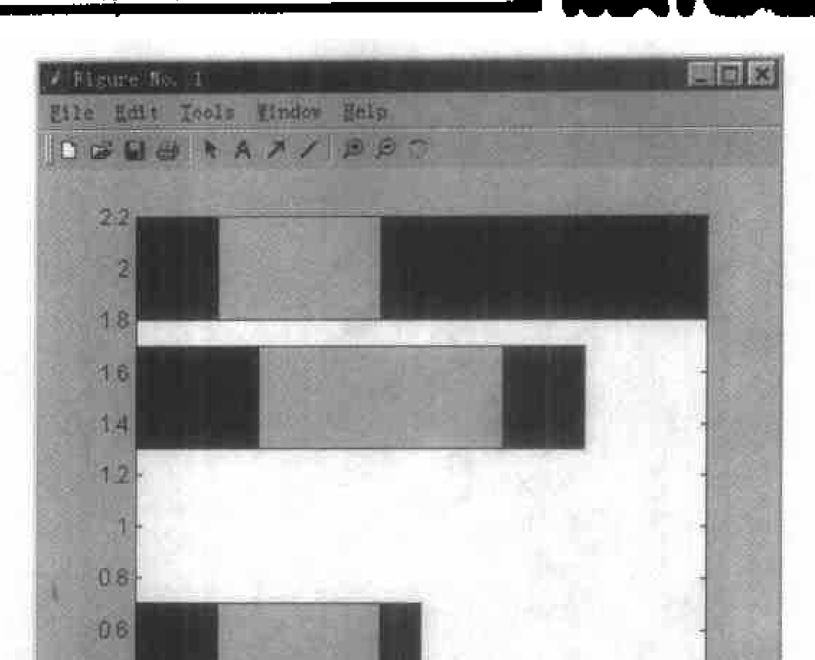


图 2-5　水平且堆叠的条形图

2.2.4　三维水平的条形图

MATLAB 中，用 bar3h 函数来绘制三维水平条形图。其格式和用法与函数 bar3 相同。只是绘出的条形图是水平的。

例程 2-6 绘制一个三维水平的条形图。

例程 2-6

```
x=[2.5 1.5 2];
y=[2 4 1;
   3 6 2;
   2 4 8];
bar3h(x,y,'group');
```

显示的结果如图 2-6 所示。

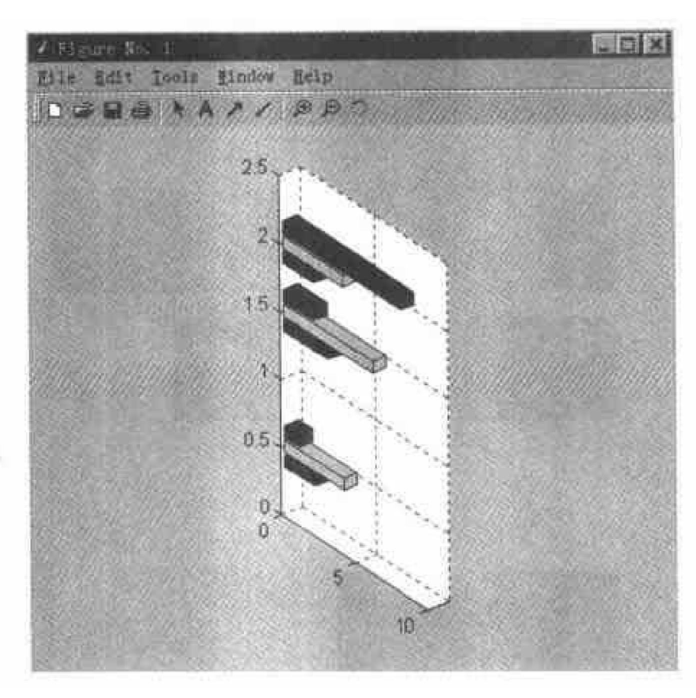


图 2-6 三维水平条形图

2.3 柱状图的绘制

柱状图是由一系列矩形条组成的，主要用于显示数据的分布情况。柱状图有笛卡儿坐标系中的柱状图和极坐标系中的柱状图。

笛卡儿坐标系中的柱状图，用矩形条的高度来代表落在一定取值范围内的数据的个数。

极坐标系中的柱状图，则用小扇形的半径来表示落在一定的角度（单位是弧度）范围内的数据的个数。

2.3.1 笛卡儿坐标系中的柱状图

在 MATLAB 中，用 hist 函数来绘制笛卡儿坐标系中的柱状图。该函数的调用格式如下：

```
n = hist(Y)
n = hist(Y,x)
n = hist(Y,nbins)
```

n = hist(Y)中的 Y 既可以是矢量，也可以是矩阵。当 Y 为矢量时，该函数自动将 *X* 轴分为 10 个等间隔区域，然后在这 10 个区域上绘制矩形条，矩形条的高度代表落在这一区域的元素的个数。当 Y 为矩阵时，则每一列都会相应地绘制一组矩形条。

n = hist(Y,x)中的 x 为 n 维矢量，则将创建 n 个区域，每个区域的中心在 x 的各个元素所在的位置上。

n = hist(Y,nbins)中的 nbins 指定了区域的个数。

所有的 n 都是用来存储落在各个区域的元素的个数。

例程 2-7 绘制一个笛卡儿坐标系中的柱状图。

例程 2-7

```
%随机生成一个 10000×3 的矩阵
Y=randn(10000,3);
hist(Y,5);
```

显示的结果如图 2-7 所示。

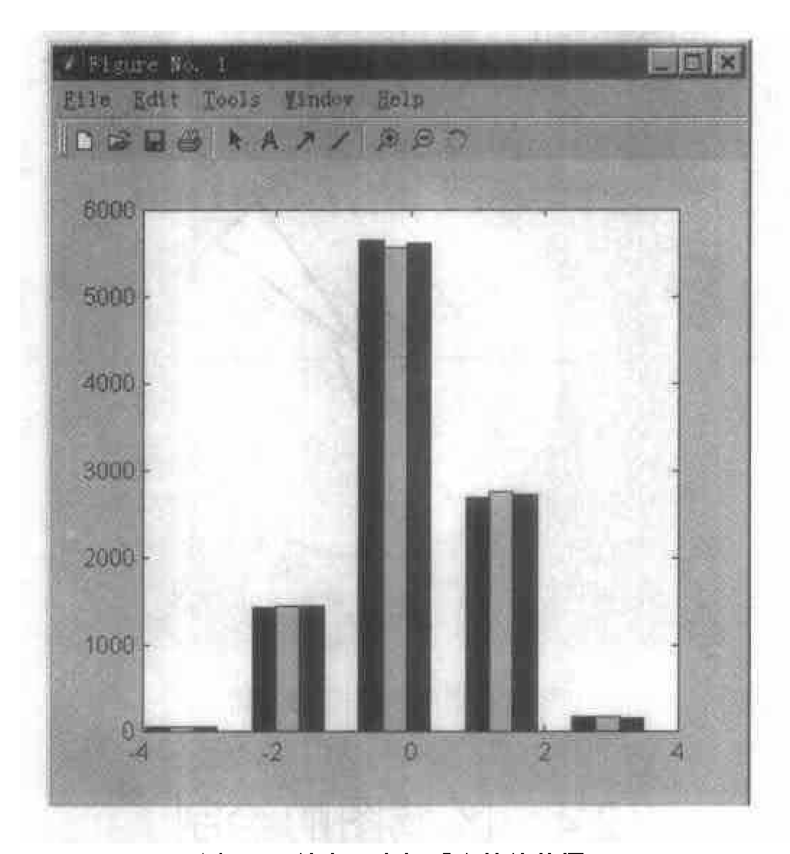


图 2-7 笛卡儿坐标系中的柱状图

2.3.2 极坐标系中的柱状图

在 MATLAB 中，用 rose 函数来绘制极坐标系中的柱状图。该函数的调用格式如下：

```
rose(theta)
rose(theta,x)
rose(theta,nbins)
```

参数 theta 为角度数据。

x 定义了小扇形的位置，它的维数决定了小扇形的个数。

nbins 定义了小扇形的个数，默认值为 20。

例程 2-8 绘制一个极坐标系中的柱状图。

例程 2-8

```
theta=[1.2 1.1 3.4 2.1 1.5 1 6 2.6 2.4 1 8];
x=2.5:2.3:3.2;
rose(theta,x);
hline=findobj(gca,'type','line');     %获取当前坐标系中的线条对象的句柄
set(hline,'linewidth',2.0);     % 将线条对象的线宽属性值设为 2.0
```

显示的结果如图 2-8 所示。

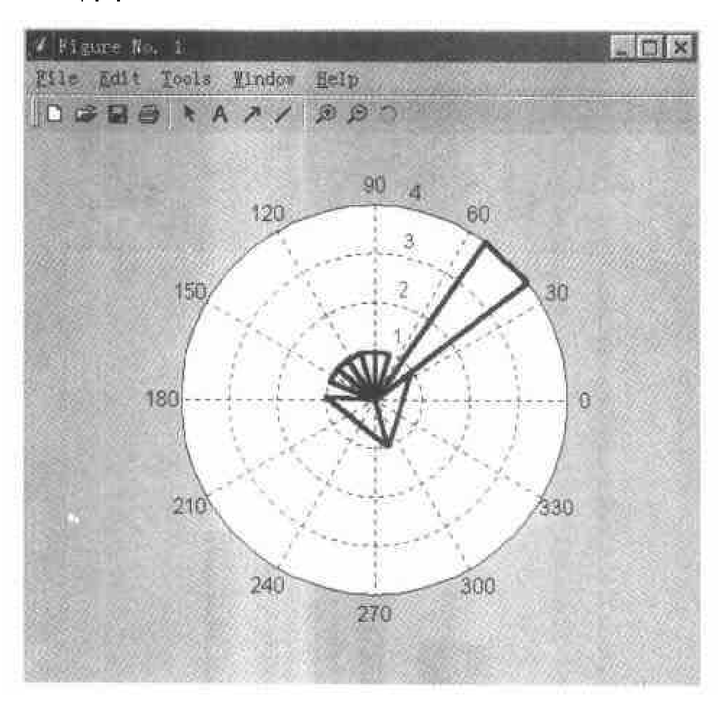


图 2-8　极坐标系中的柱状图

2.4　区域图的绘制

MATLAB 中，用函数 area 来绘制区域图。该函数的调用格式如下：

```
area(Y)
area(X,Y)
area(…,ymin)
area(…,'PropertyName',PropertyValue, …)
h = area(…)
```

area(Y)中的 Y 既可以是矢量也可以是矩阵。当 Y 为矢量时，该函数先绘制一条连接(Y(i),I)的折线，然后对该曲线与 X 轴之间的区域进行填充；当 Y 为矩阵时，则针对该矩阵的每一列都绘制一条曲线，然后对曲线与 X 轴和曲线与曲线之间的区域进行填充。需要注意的示，对应第 i 列的折线上的点坐标为((Y(j,i)+Y(j,i-1)),j)，即要加上前一列所对应的元素。

area(X,Y)所表示的意义与 area(Y)类似，只不过折线上的点坐标的 x 坐标由 X 来确定。

area(…,ymin)中的 ymin 表示填充的区域为所绘制折线与 y=ymin（默认情况下为 y=0，即 *X* 轴）之间的区域。

Area（…,'PropertyName',PropertyValue, …)表示用户可以设置特定的对象属性，来控制区域图显示的外观。

例程 2-9 创建区域对象。

例程 2-9

```
Y=[3 1 2
   8 7 5
   5 3 4
   9 3 2];
area(Y)
grid on
set(gca,'layer','top');
```

显示的结果如 2-9 所示。

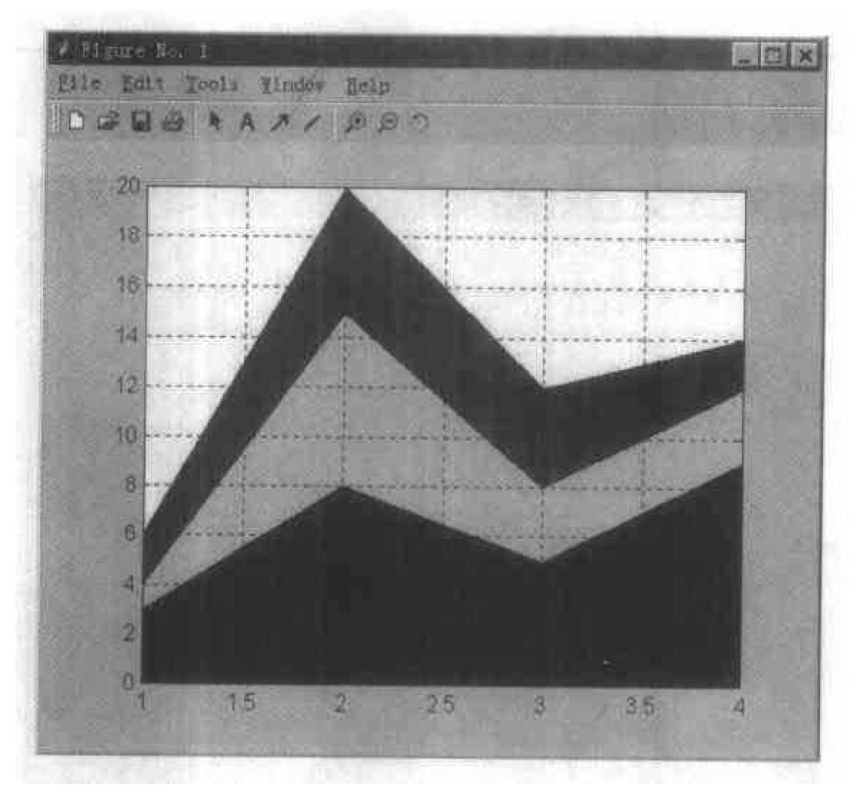


图 2-9　由矩阵数据绘制的区域图

2.5　饼图的绘制

在 MATLAB 中，饼图用于显示矢量或矩阵中的每个元素在其所有的总和中所占的百分比。

MATLAB 中绘制饼图的函数有两个，即 pie 函数和 pie3 函数，它们分别用于创建二维和三维饼图。下面就分别介绍这两个函数的用法。

2.5.1 二维饼图的绘制

在 MATLAB 中，用 pie 函数来实现二维饼图的绘制。该函数的调用格式如下：

```
pie(x)
h=pie(x,explode)
h=pie(x,labels)
```

pie(x)中，x 既可以是矩阵也可以是矢量。且如果 x 中的各个元素的和大于或等于 1 时，则饼图中的每个切片占整个饼图的百分比为 x(i,j)/sum。如果 x 中的各个元素的和小于 1 时，则绘制的是一个不完全饼图，且每个切片占整个饼图的百分比为 x(i,j)/1=x(i,j)。

例程 2-10 绘制一个不完全饼图

例程 2-10

```
x=[2.1,2.3,2.4,2.1];
pie(x);
```

显示结果如图 2-10 所示。

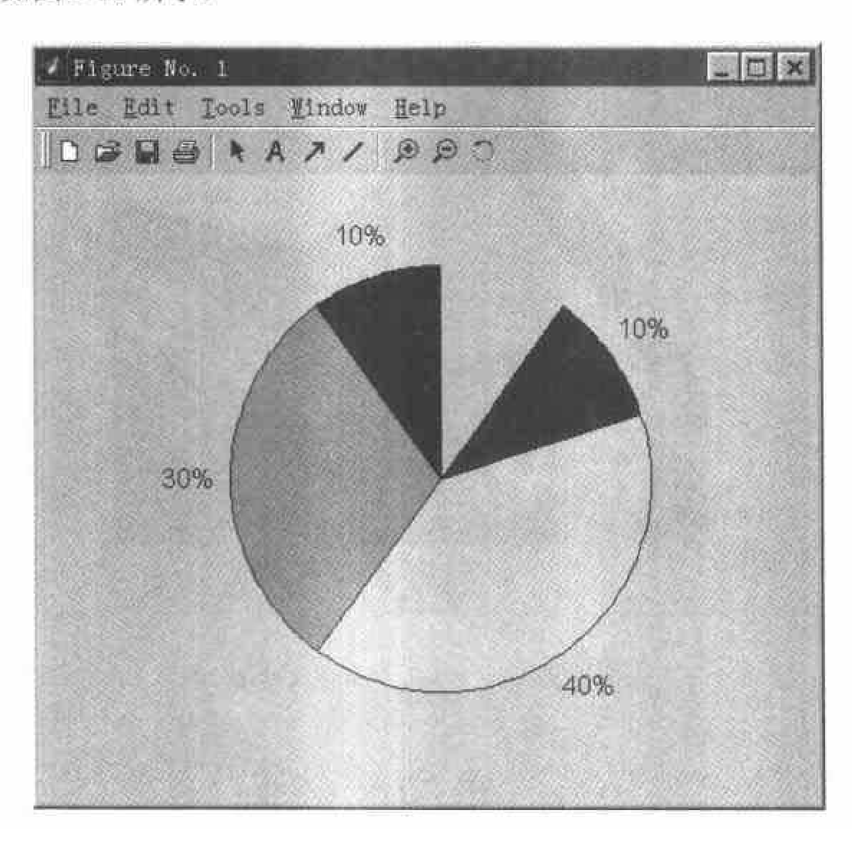
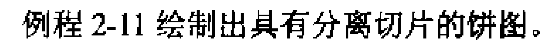


图 2-10 不完全饼图



h=pie(x,explode)用来绘制具有分离切片的饼图。参数 explode 必须与 x 的维数相同，explode 中非零元素所对应的切片即为应分离的切片。



例程 2-11 绘制出具有分离切片的饼图。

例程 2-11

```
x=[1,2,3,4];
explode=[0,0;1,1];
pie(x,xplode);
```

显示的结果如图 2-11 所示。

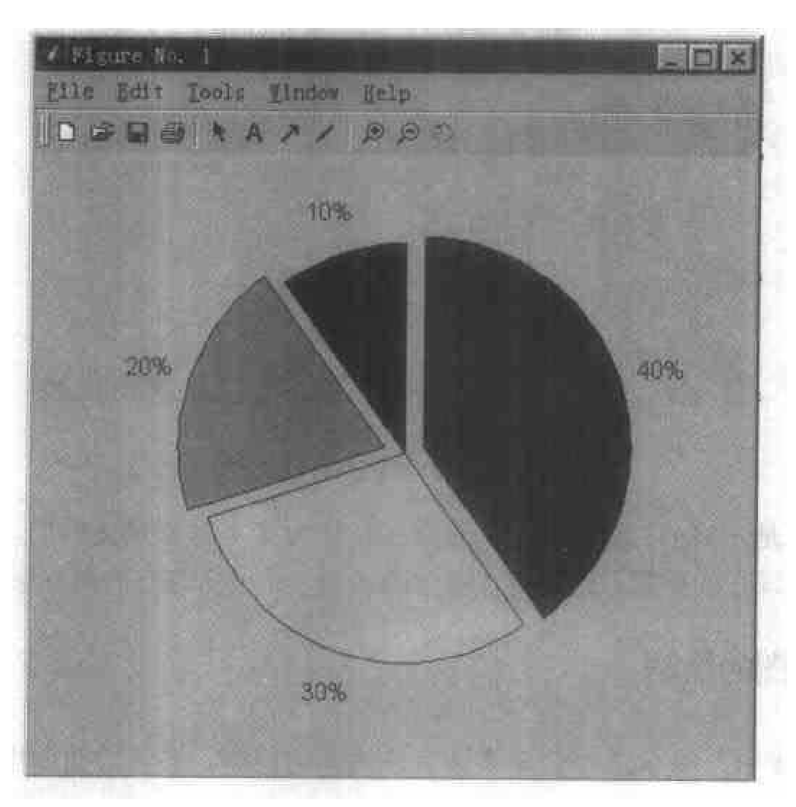


图 2-11　带分离切片的饼图



h=pie(x,labels)用来绘制具有特定标注的饼图。参数 labels 必须与 x 的维数相同，且它里面包含的元素只能是字符串。

例程 2-12 绘制出具有特定标注的饼图。

例程 2-12

```
x=[80 81 70 110];
labels={'一队'  '二队'  '三队'  '四队'};
pie(x,labels);
```

显示的结果如图 2-12 所示。

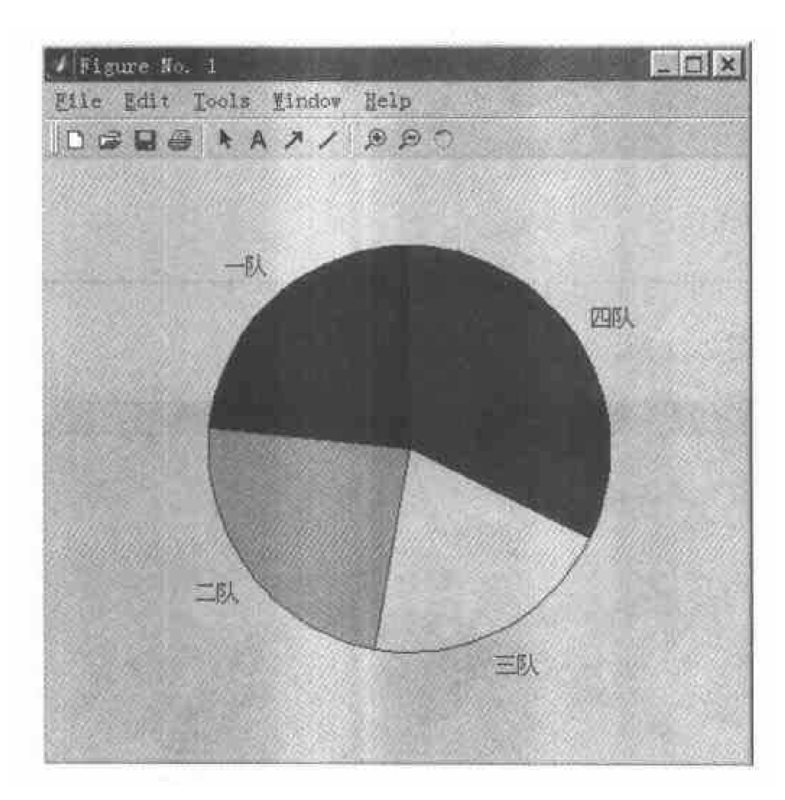


图 2-12 带有特定标注的饼图

需要注意的是，MATLAB 中饼图的标注是作为文本对象来处理的，所以如果要修改标注文本字符串或它的位置，首先要获取它的句柄，然后修改它的属性来实现上述的目的。

2.5.2 三维饼图的绘制

在 MATLAB 中，用 pie3 函数来实现三维饼图的绘制。该函数的调用格式如下：

```
pie3(x)
h=pie3(x,explode)
h=pie3(x,labels)
```

其中，各项参数的意义与 pie 函数中各项参数的意义完全相同，此处不再赘述。

2.6 离散数据的图形绘制

工程应用和科学计算中经常会得到许多离散数据，MATLAB 提供了一些特殊的函数来将这些离散数据以图形的形式显示出来，使用户能够更加直观地看出这些离散数据的内在特点和规律。显示离散数据的图形有：

- 二维枝干图
- 三维枝干图
- 阶梯图

下面一一对其进行介绍。

2.6.1　二维枝干图

枝干图即将每个离散数据显示为末端带有标记符号的线条（即枝干）。在二维枝干图中，枝干线条的起点在 X 坐标轴上。

MATLAB 中，用 stem 函数来绘制二维枝干图。其调用格式如下：

```
stem(x,y,'linestyle or marker or color','ill')
```

参数 x 和 y 表示要绘制的离散数据。

参数'linestyle or marker or color'表示用户可以自定义枝干的线型、标记符号的类型和颜色。

参数'fill'表示对于有些标记符号，我们还可以对其进行填充。

例程 2-13 绘制出一个二维枝干图。

例程 2-13

```
%线型被指定为虚线（用“..”表示），标记符号为菱形（用“d”表示），线条和
%标记符号的颜色为蓝色（用“b”表示）。
x=0:2.2:2;
y=sin(x);
stem(x,y,'..db','fill');
```

显示的结果如图 2-13 所示。

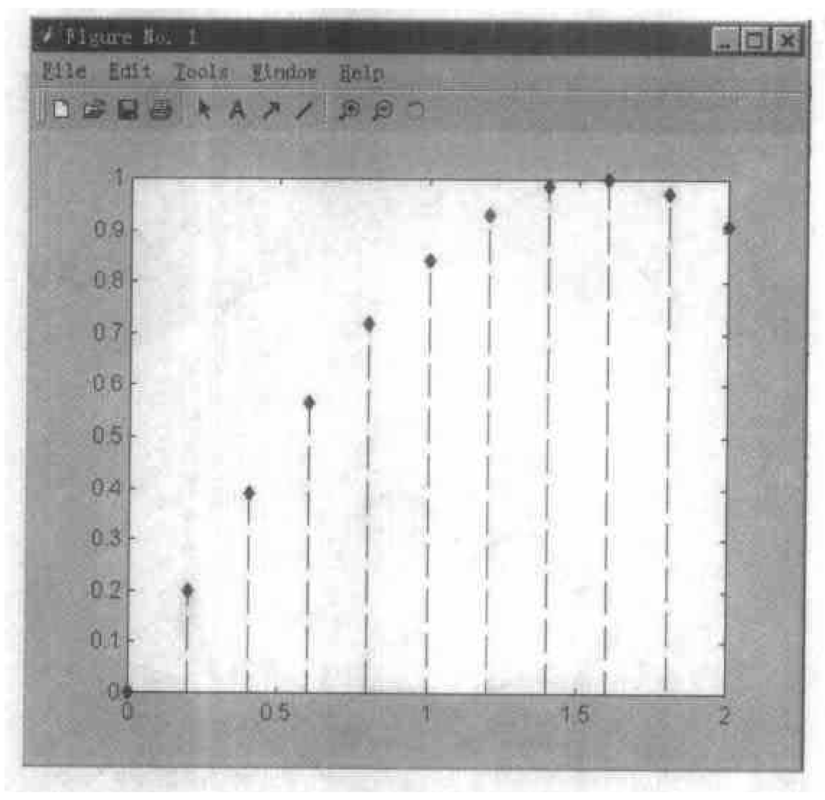


图 2-13　二维枝干图

2.6.2 三维枝干图

MATLAB 中，用 stem3 函数绘制起点在 *XY* 平面上的三维枝干图。其调用格式如下：

stem3(z)

stem3(x,y,z,'linestyle or marker or color','fill')

stem3(z)表示将只在 x=1（当 z 为一个列矢量时）或 y=1（当 z 为一个行矢量时）处绘制一行枝干图。

第二种格式中的各项参数与二维枝干图的解释类似。

例程 2-14 利用三维枝干图显示快速傅里叶变换的计算过程。

例程 2-14

```
th=(0:127)/128*2*pi;
x=cos(th);
y=sin(th);      %计算复平面上的单位圆
f=(abs(fft(ones(10,1),128)))';  % 计算一步频率响应的幅值
stem3(x,y,f,'*');      %绘制三维枝干图
xlabel('实部');
ylabel('虚部');
zlabel('幅值');
title('频率响应');      %进行标注
```

显示的结果如下图 2-14 所示。

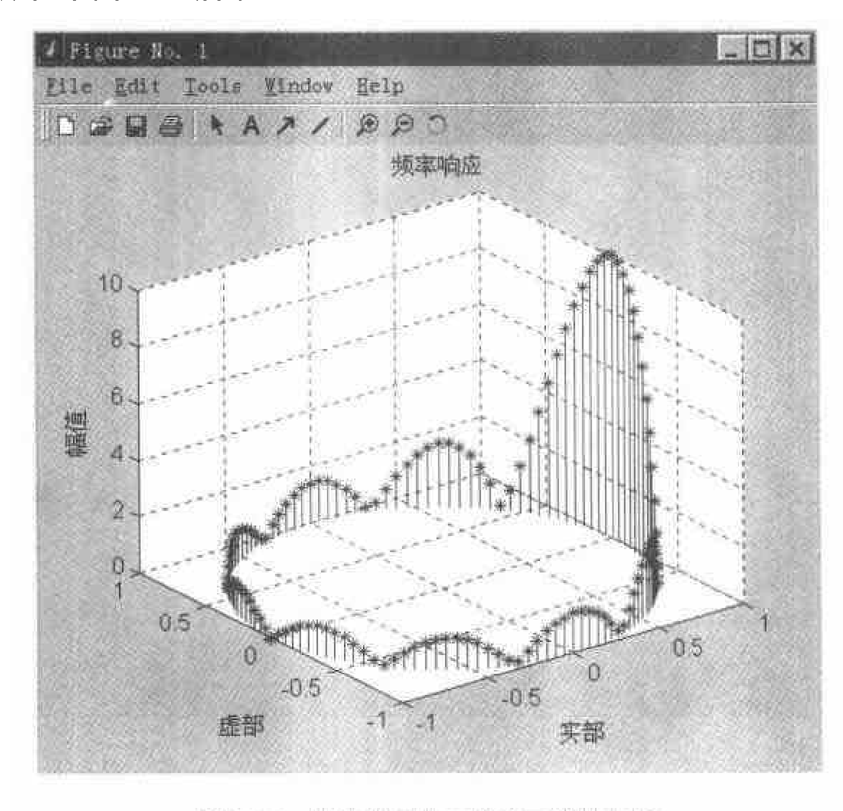


图 2-14 快速傅里叶变换的三维枝干图

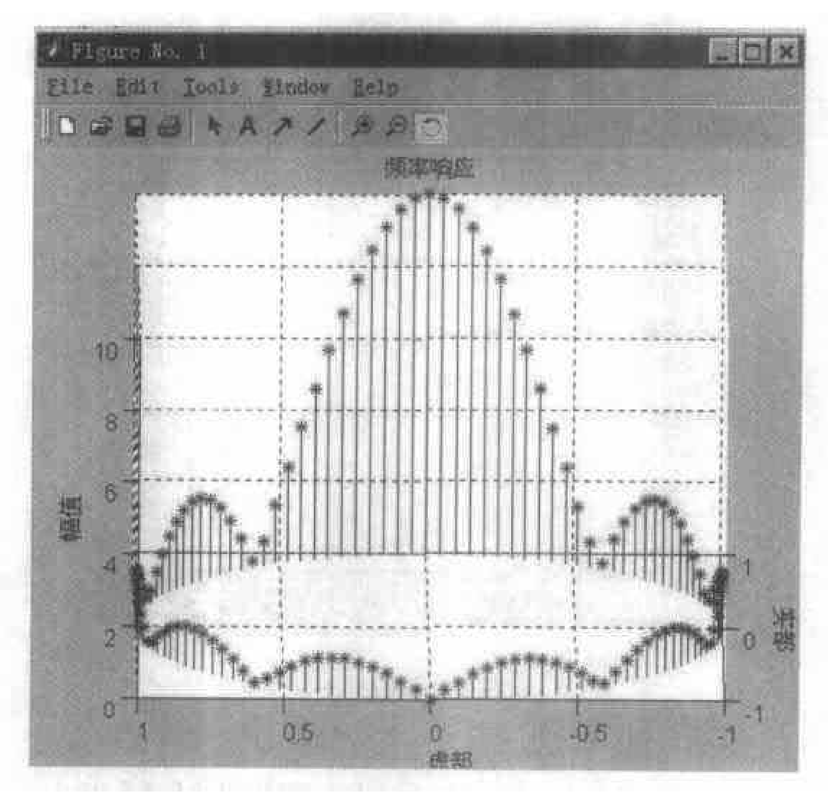


图 2-15　旋转后的三维枝干图

如果需要换个角度查看三维枝干图，则首先执行下面的代码：

```
rotate3d on
```

然后就可以用鼠标拖动该三维枝干图，转到希望的角度。图 2-15 给出了旋转后的该三维枝干图。

2.6.3　阶梯图

阶梯图以一个恒定间隔的上升沿显示数据。在阶梯图中，在 X 坐标轴的 $x(i)$到 $x(i+1)$之间的所有位置的 y 值都为 $y(i)$。

MATLAB 中，用函数 stairs 来绘制阶梯图。其调用格式如下：

```
stairs(x,y,'linestyle or marker or color')
```

其中，参数'linestyle'用于指定绘制阶梯的线形、标记符号及颜色。

例程 2-15 绘制一个用户指定线形和标记符号及颜色的阶梯图。

例程 2-15

```
x=1:10;
y=sin(x);
stairs(x,y,':rd');
```

显示的结果如图 2-16 所示。

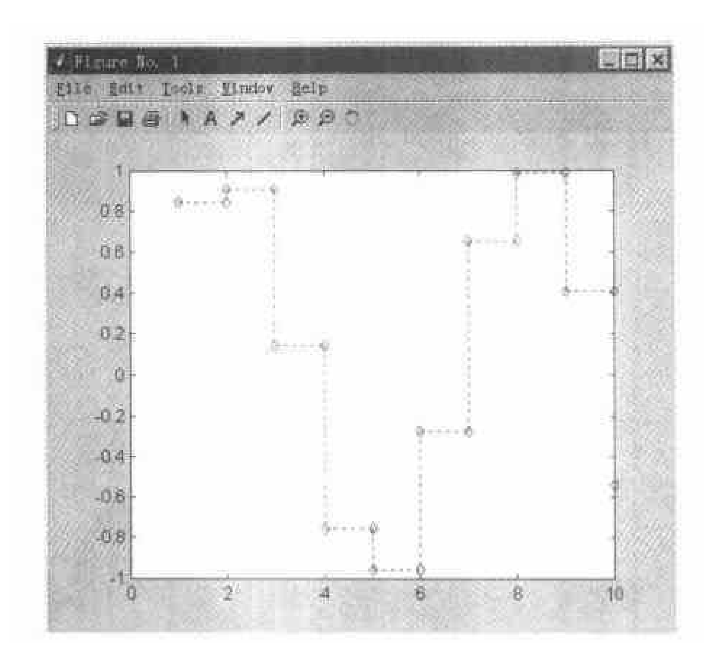


图 2-16 阶梯图

2.7 方向和速度矢量图的绘制

在 MATLAB 中，提供了一些特殊的函数，专门用于绘制显示方向矢量或速度矢量数据的图形。这些图形包括：

- 罗盘图
- 羽状图
- 二维箭头图
- 三维箭头图

2.7.1 罗盘图

罗盘图用来显示起点在图形原点的矢量。MATLAB 中，用函数 compass 来绘制罗盘图。该函数的调用格式如下：

```
compass(x)
compass(x,y,'linespec')
```

compass(x,y,'linespec')表示绘制以图形原点为起点，以（x(i),y(i)）为终点的矢量箭头图。最后一个参数表示用户可以自定义箭头矢量的绘制特征（如线型、颜色等）。

compass(x)则等价于 compass(real(x),iamg(x))。

需要注意的是，该函数采用的是笛卡儿坐标系，而且是在圆形栅格上绘制图形。但圆形栅格的最外层标注的却是在极坐标系下的角度，各个小圆上标注的则是矢径的长度。所以当给定极坐标时，要先将其转换为直角坐标（用 pol2cart 函数即可），再绘制罗盘图。

例程 2-16 绘制风向与风力的罗盘图。

例程 2-16

```
w_direction=[45 90 90 45 180];
w_intensity=[6 7 3 8 5];    %以上为代表风向和风力的矢量

arcw_direction=w_direction*pi/180;
[x,y]=pol2cart(arcw_direction,w_intensity);  %将极坐标转换为直角坐标
compass(x,y);          %绘制罗盘图
title('长沙地区五小时内的风向和风力图');  %标注
```

显示的结果如图 2-17 所示。

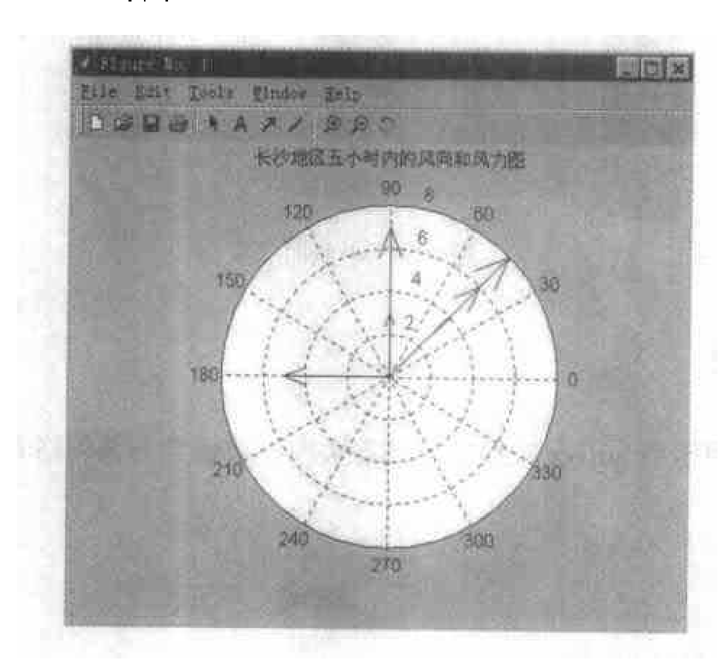


图 2-17　风向和风力的罗盘图

2.7.2　羽状图

羽状图也是以箭头的形式显示一系列矢量，与罗盘图不同的是，这些矢量的起点不是在原点，而是均匀的分布在一条与 X 轴平行的直线上。

MATLAB 中，用函数 feather 来绘制羽状图。该函数的调用格式如下：

feather(x)

feather(x,y,'linespec')

关于这两种格式的说明与在罗盘图的类似，此处不再赘述。

例程 2-17 绘制羽状图。

例程 2-17

```
x=[3 4 5 6];
y=[2 3 4 5];
feather(x,y);
```

显示的结果如图 2-18 所示。

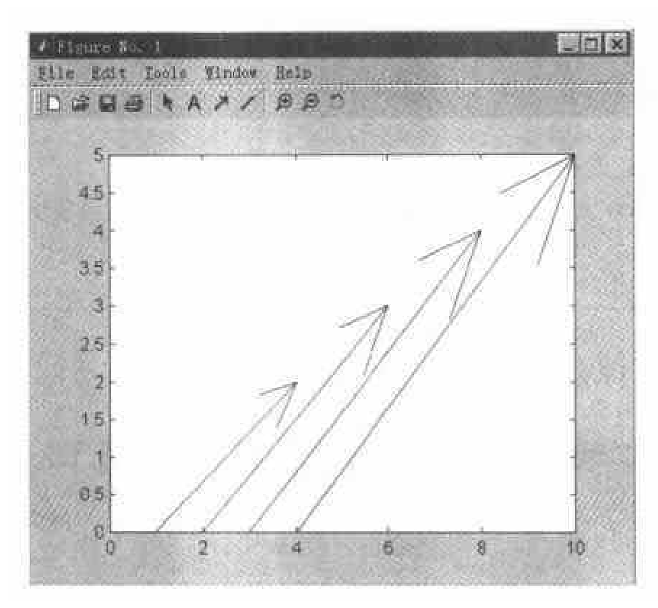


图 2-18　羽状图

2.7.3　二维箭头图

MATLAB 中，用函数 quiver 来绘制二维箭头图。该函数的调用格式如下：

```
quiver(U,V)
quiver(X,Y,U,V)
quiver(…,scale)
quiver(…,LineSpec)
quiver(…,LineSpec,'filled')
h = quiver(…)
```

X 和 Y 中的元素组成的点（x,y）作为起点，U 和 V 中的元素组成的点（u,v）作为终点，且其数值是相对起点而言的。

scale 代表缩放比例，为 0 时，矢量是原长度；默认值为 1，此时自动调整缩放比例，以防止交叠；为 2.5 时，长度为 1 时的一半。

例程 2-18　绘制一个二维箭头图。

例程 2-18

```
x=[1 2 3];
y=[1 2 3];
u=[7 8 9];
v=[4 5 6];
quiver(x,y,u,v,0);
```

显示的结果如图 2-19 所示。

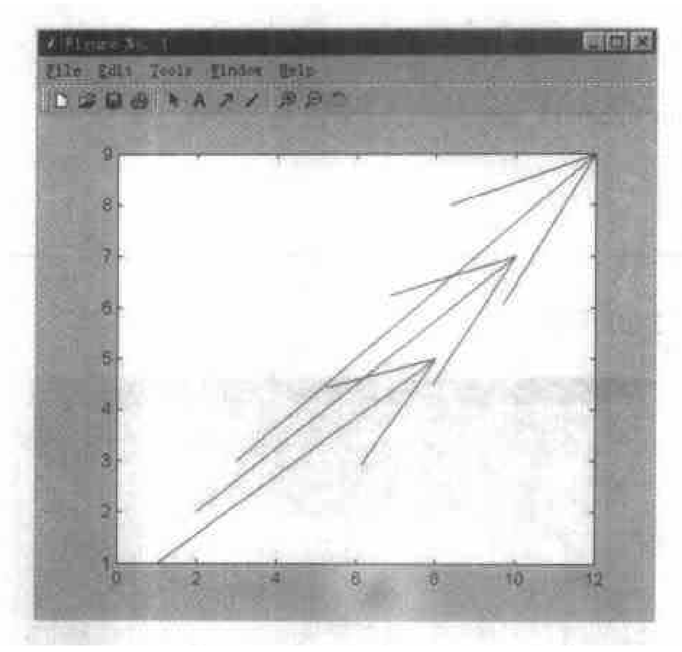


图 2-19 二维箭头图

2.7.4 三维箭头图

MATLAB 中，用函数 quiver 来绘制三维箭头图。该函数的调用格式如下：

```
quiver3(Z,U,V,W)
quiver3(X,Y,Z,U,V,W)
quiver3(…,scale)
quiver3(…,LineSpec)
quiver3(…,LineSpec,'filled')
h = quiver3(…)
```



其中，各项参数的说明与二维箭头图的类似，只是在三维坐标系中多了个 z 坐标。

例程 2-19 用三维箭头图显示抛物线。

例程 2-19

```
vz=10;
a=.32;
t=0:2.1:1;
z=vz*t+1/2*a*t.^2;
vx=2;
x=vx*t;
vy=3;
y=vy*t;
u=gradient(x);
v=gradient(y);
```

```
w=gradient(z);
scale=0;
quiver3(x,y,z,u,v,w,scale)
xlabel('x 轴');
ylabel('y 轴');
zlabel('z 轴');
```

显示的结果如图 2-20 所示。

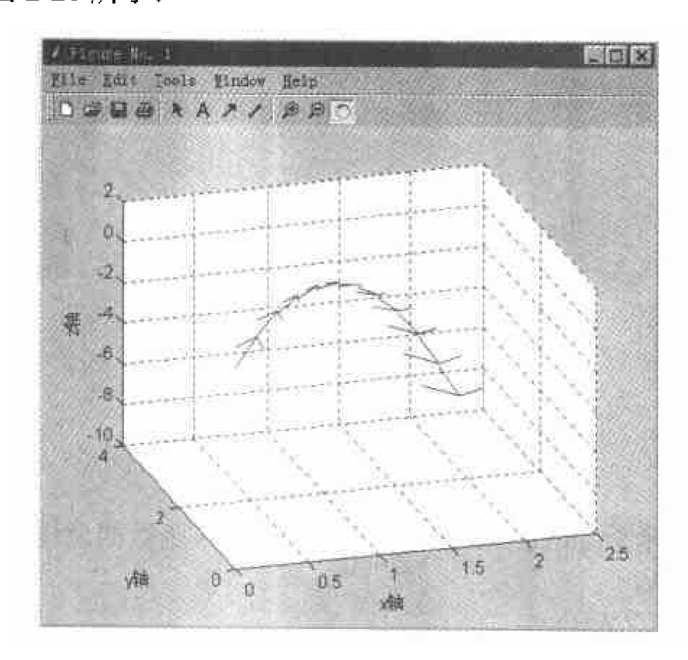


图 2-20　三维箭头图

2.8　轮廓图的绘制

MATLAB 中的轮廓图即指将相对于某一平面（即 *XY* 平面）具有同一高度的点连成一条曲线，而该高度是通过高度矩阵来反映的。在 MATLAB 中有两类轮廓图，即二维轮廓图和三维轮廓图。

2.8.1　二维轮廓图

MATLAB 中，用函数 contour 来绘制二维轮廓图，该函数的调用格式如下：

```
contour(Z)
contour(Z,n)
contour(Z,v)
contour(X,Y,Z)
contour(X,Y,Z,n)
contour(X,Y,Z,v)
contour(…,LineSpec)
[C,h] = contour(…)
```



Z即为高度矩阵，X和Y代表X轴和Y轴的取值，且X和Y的维数与Z的维数必须匹配。若X和Y是矩阵，则必与Z的维数相同，此时将绘制一个表面对象。

参数n代表轮廓线的条数。

参数v为一个矢量，用来定义在何处绘制轮廓线，各个元素的取值即代表了轮廓线的基准值。

[C,h] = contour(…)表示返回一个轮廓矩阵C和一个图形句柄矢量h。利用这两个参数，可调用函数clabel对轮廓图进行标注。

例程2-20 绘制一个具有标注的二维轮廓图。

例程 2-20

```
[X,Y,Z]=peaks;
[C,h]=contour(X,Y,Z,15,'r*');
clabel(C,h);
title('具有15条轮廓线的peaks函数');
```

显示的结果如图2-21所示。

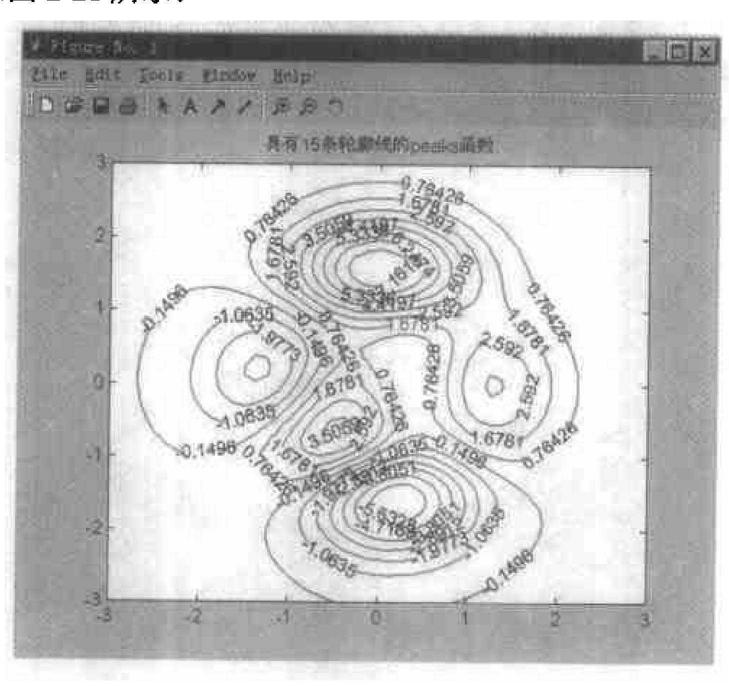


图2-21 带标注的二维轮廓图

2.8.2 三维轮廓图

MATLAB中，用函数contour3来绘制二维轮廓图。该函数的调用格式如下：

```
contour3(Z)
contour3(Z,n)
contour3(Z,v)
contour3(X,Y,Z)
contour3(X,Y,Z,n)
```

```
contour3(X,Y,Z,v)
contour3(…,LineSpec)
[C,h] = contour3(…)
```

关于各项参数的说明与 contour 函数相同。

例程 2-21 绘制一个具有标注的三维轮廓图。

例程 2-21

```
[X,Y,Z]=peaks;
[C,h]=contour3(X,Y,Z,15,'r*');
title('具有 15 条轮廓线的 peaks 函数');
xlabel 'x';
ylabel 'y';
zlabel 'z';
```

显示的结果如图 2-22 所示。

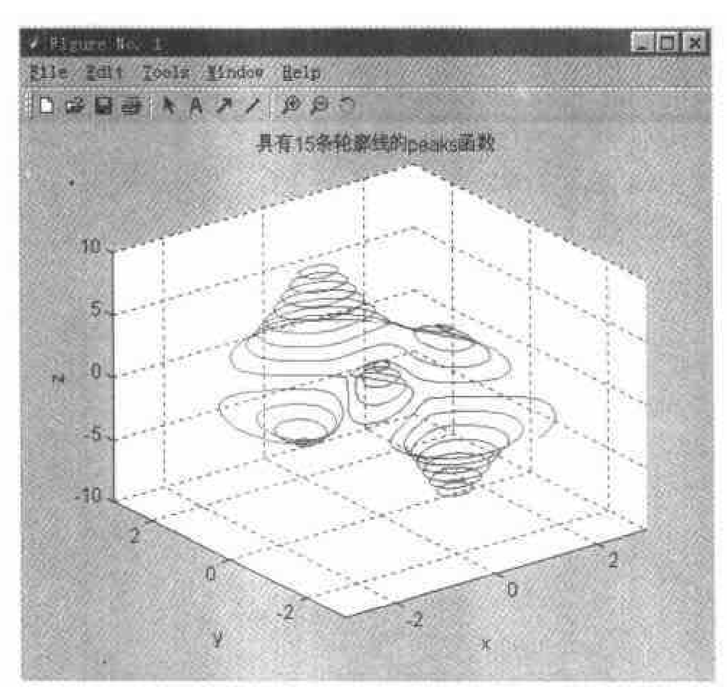


图 2-22 三维轮廓图

2.9 动画的绘制

MATLAB 中，用两种方法来创建动画：

- 首先保存一系列图形，然后按照一定的顺序像电影一样播放。这种方式的动画称之为电影动画。
- 在图形窗口中按照一定的算法连续擦除和重绘图形对象。这种方式的动画称之为程序动画。



2.9.1　电影动画

MATLAB 中，创建电影动画的过程分为以下三步：

（1）调用 moviein 函数对内存进行初始化，创建一个足够大的矩阵，使之能够容纳基于当前坐标轴大小的一系列指定的图形（此处称为帧）。

（2）调用 getframe 函数生成每个帧。该函数返回一个列矢量，利用这个矢量，就可以创建一个电影动画矩阵。

（3）调用 movie 函数按照指定的速度和次数运行该电影动画。

例程 2-22 创建一个电影动画，用来演示快速傅里叶变换的过程。

例程 2-22

```
axis equal;
M=moviein(16,gcf);
set(gca,'nextplot','replacechildren');
h=uicontrol('style','slider','position',...
[100 10 500 20],'min',1,'max',16);
for j=1:16
    plot(fft(eye(j+16)))
    set(h,'value',j)
    M(:,j)=getframe(gcf);
end
clf
axes('position',[0 0 1 1]);
movie(M,30)
```

动画播放过程中的一个画面如图 2-23 所示。

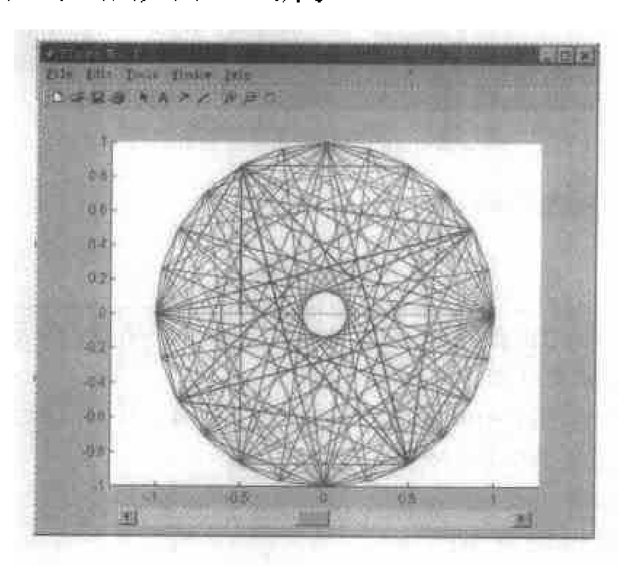


图 2-23　电影动画的播放画面

2.9.2　程序动画

在 MATLAB 中，采用重绘图形的方法来创建程序动画。改变对象的方法可以触发 MATLAB 对该对象进行重绘。利用图形对象的这种属性，创建程序动画的典型步骤是：

（1）绘制一个图形对象。

（2）在程序循环中改变该对象的 *x*、*y*、*z* 坐标值来实现对象的移动，从而形成动画。

另外，在创建 MATLAB 的程序动画时，图形的擦除也是很重要的。通过设置不同的擦除模式，用户可以得到各种动画效果。MATLAB 为用户提供了 3 种擦除模式：

- None：在移动图形对象时，不进行擦除。
- Background：在图形对象移走后，在原来的位置用背景色进行重绘。在这种模式下，MATLAB 将原来的对象完全擦除，包括该对象下面的所有图形。
- Xor：与第二种模式相比，这种模式只擦除对象本身。大多数的 MATLAB 程序动画都采用这种擦除模式。

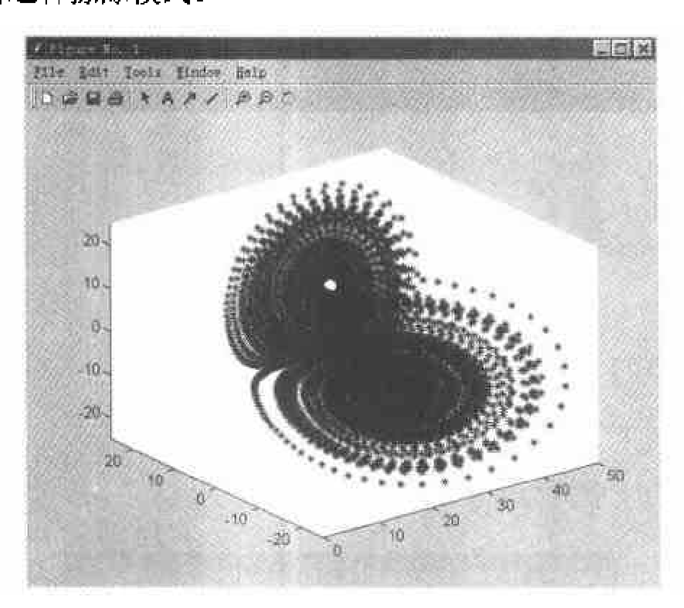


图 2-24　不进行擦除的程序动画

例程 2-23 创建 ·个程序动画。

例程 2-23

```
%在二维空间演示一个由著名的劳伦兹非线性奇异方程 所描述的无序运动。先介绍
%下劳伦兹方程，其形式如下：dy/dt=Ay
%方程中包含 一个由函数 y(t)确定的矢量和一个矩阵 A。且 A 依赖于 y，存在着下面的
%关系：A=[.8/3 ,0, y(2);0,.10,10;.y(2),28,.1]
%该程序的代码如下：
A=[.8/3 0 .10;0 .10 10;10 28 .1];
y=[35;.10;.7];
h=2.01;
p=plot3(y(1,1),y(2,1),y(3,1),'*','erasemode','none');
axis([0 50 .25 25 .25 25]);
```

```
    hold on;
    for i=1:8000
        A(1,3)=y(2,1);
        A(3,1)=-y(2,1);
        dita_y=h*A*y;
        y=y+dita_y;
set(p,'xdata',y(1,1),'ydata',y(2,1),'zdata',y(3,1));
        drawnow;
    end
```

程序运行的结果如图 2-24 所示。

在上面的程序中由于我们把擦除模式设置为了'none'，所以我们在图形窗口中看到的是由于点的移动所留下的轨迹。

如果将擦除模式设置为'background'，那么我们看到的只是一个点在坐标系中的运动所形成的动画，动画播放过程中的一个画面如图 2-25 所示。

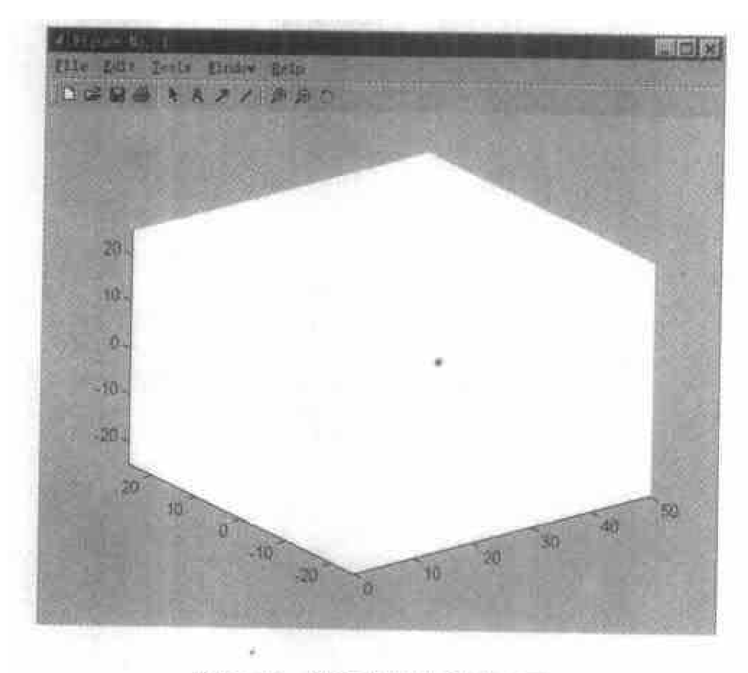


图 2-25　进行擦除的程序动画

第3章　MATLAB图像及其操作

进行图像处理前，首先要很好地了解一些关于图像的基础知识。本章将针对MATLAB中有关图像的一些最基本的知识进行较为系统的介绍，这些知识包括图像的文件格式、图像的类型及图像的存储类型等。

本章为后续各章的基础，建议读者仔细阅读。

本章主要内容：

- MATLAB图像文件格式
- MATLAB图像类型
- MATLAB图像文件的读写
- MATLAB图像对象属性

3.1　MATLAB图像文件格式

MATLAB支持以下几种图像文件格式：

- PCX（Windows Paintbrush）格式。可处理1、4、8、16、24位等图像数据。文件内容包括：文件头（128字节）、图像数据、扩展颜色映射表数据。
- BMP（Windows Bitmap）格式。有1、4、8、24位非压缩图像，8位RLE（Runlength Encoded）图像。文件内容包括：文件头（一个数据结构）、位图信息数据块（位图信息头和一个颜色表）和图像数据。
- HDF（Hierarchical Data Format）格式。有8位、24位光栅图像数据集。
- JPEG（Joint Photographic Experts Group）格式。是一种称为联合图像专家组的图像压缩格式。
- TIFF（Tagged Image File Format）格式。处理1、4、8、24位非压缩图像，1、4、8、24位packbit压缩图像，1位CCITT压缩图像等。文件内容包括：文件头、参数指针表与参数域、参数数据表和图像数据4部分。
- XWD（X Windows Dump）格式。包括1、8位Zpixmaps，XYBitmaps，1位XYPixmaps。
- PNG（Portable Network Graphics）格式。

3.2　图　像　类　型

MATLAB中，一幅图像可能包含一个数据矩阵，也可能包含一个颜色映射表矩阵。MATLAB中有4种基本的图像类型：

- 索引图像

- 灰度图像
- RGB 图像
- 二进制图像

此外，MATLAB 还支持由多帧图像组成的图像序列。下面具体介绍 MATLAB 如何描述这几种图像。

3.2.1 索引图像

索引图像包括一个数据矩阵 ***X***，一个颜色映像矩阵 ***Map***。其中 ***Map*** 是一个包含三列和若干行的数据阵列，其每一个元素的值均为[0，1]之间的双精度浮点型数据。***Map*** 矩阵的每一行分别表示红色、绿色和蓝色的颜色值。在 MATLAB 中，索引图像是从像素值到颜色映射表值的“直接映射”。像素颜色由数据矩阵 ***X*** 作为索引指向矩阵 ***Map*** 进行索引。例如，值 1 指向矩阵 ***Map*** 中的第一行，2 指向第二行，依此类推。

颜色映射表通常和索引图像存在一起。当用户在调用函数 imread 时，MATLAB 自动将颜色映射表与图像同时加载。在 MATLAB 中可以选择所需要的颜色映射表，而不必局限于使用默认的颜色映射表。我们可以使用属性 CDataMapping 来选取其他的颜色映射表，包括用户自定义的颜色映射表。

图 3-1 显示了索引图像的结构。该图像中的像素用整数类型表示，指向颜色映射表中的特定的颜色值。

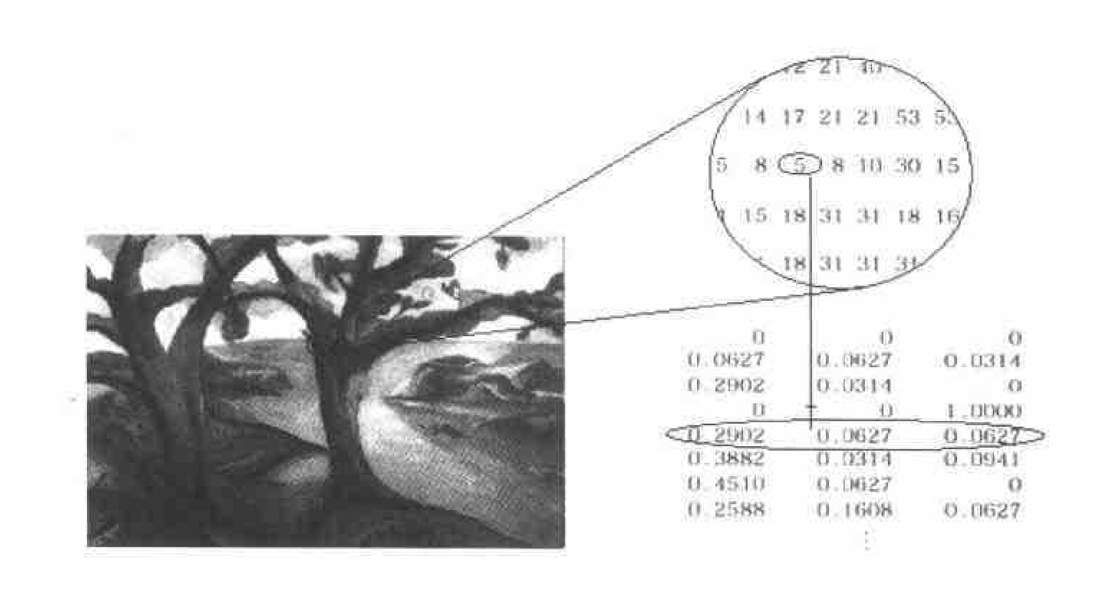


图 3-1 索引图像的结构

需要说明的是，图像矩阵与颜色映射表之间的关系取决于图像数据矩阵的类型。如果图像数据矩阵是双精度的数据类型，则值 1 指向矩阵 ***Map*** 中的第一行，值 2 指向第二行，等等；如果图像矩阵是 8 位无符号的整数类型或 16 位无符号的整数类型，则由于存在一个偏移量，所以值 0 指向矩阵 ***Map*** 中的第一行，值 1 指向第二行，依此类推。在图 3-1 中，图像矩阵用的是双精度型，无偏移量，所以值 5 指向颜色映射表中的第 5 行。

例程 3-1 显示一幅索引图像。

例程 3-1

```
[X,map]=imread('canoe.tif');
image(X);
colormap(map)
```

3.2.2 灰度图像

MATLAB 中，一幅灰度图像是一个数据矩阵 *I*，其中 *I* 中的数据均代表了在一定范围内的颜色灰度值。MATLAB 把灰度图像存储为一个数据矩阵，该数据矩阵中的元素分别代表了图像中的像素。矩阵中的元素可以是双精度的浮点数类型、8 位或 16 位无符号的整数类型。大多数情况下，灰度图像很少和颜色映射表一起保存。但是在显示灰度图像时，MATLAB 仍然在后台使用系统预定义的默认的灰度颜色映射表。

图 3-2 显示了灰度图像的结构。

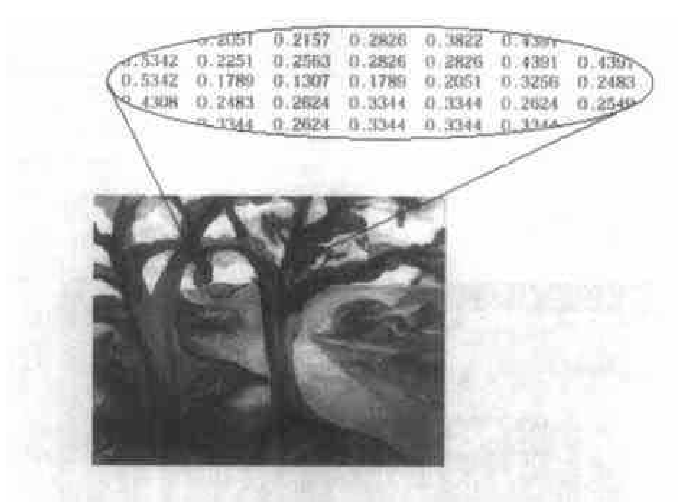


图 3-2　灰度图像的结构

在 MATLAB 中，要显示一幅灰度图像，需要调用函数 imagesc（即 image scale，图像缩放函数）。

例程 3-2 用具有两个输入参数的 imagesc 函数显示一幅灰度图像。

例程 3-2

```
I=imread('moon.tif');
imagesc(I,[0 1]);
colormap(gray)
```

imagesc 函数中的第二个参数确定灰度范围。灰度范围中的第一个值（通常是 0）对应与颜色映射表中的第一个值（颜色），灰度范围中的第二个值（通常是 1）对应与颜色映射表中的最后一个值（颜色）。在灰度范围中间的直线型对应于颜色映射表中的剩余的值（颜色）。

3.2.3 RGB 图像

RGB 图像，即真彩图像，在 MATLAB 中存储为 $n \times m \times 3$ 的数据矩阵。数组中的元素定义了图像中每一个像素的红、绿、蓝颜色值。需要指出的是，RGB 图像不使用 Windows 颜色映射表。像素的颜色由保存在像素位置上的红、绿、蓝的灰度值的组合来确定。图形文件格式把 RGB 图像存储为 24 位的图像，红、绿、蓝分别占 8 位。这样可以有约一千万种颜色（即 2^{24}=16777213）。

MATLAB 的 RGB 数组可以是双精度的浮点数类型、8 位或 16 位无符号的整数类型。在 RGB 的双精度型数组中，每一种颜色是用在 0 和 1 之间的数值表示。例如，颜色值是（0,0,0）的像素，显示的是黑色；颜色值（1,1,1）的像素，显示的是白色。每一像素的三个颜色值保存在数组的第三维中。例如，像素（10,5）的红、绿、蓝颜色值分别保存在元素 RGB(10,5,1)、RGB(10,5,2)、RGB(10,5,3)中。

例程 3-3 调用函数 image 来显示 RGB 图像。

例程 3-3

```
RGB=imread('flowers.tif');
image(RGB)
```

打开的真彩图像如图 3-3 所示。

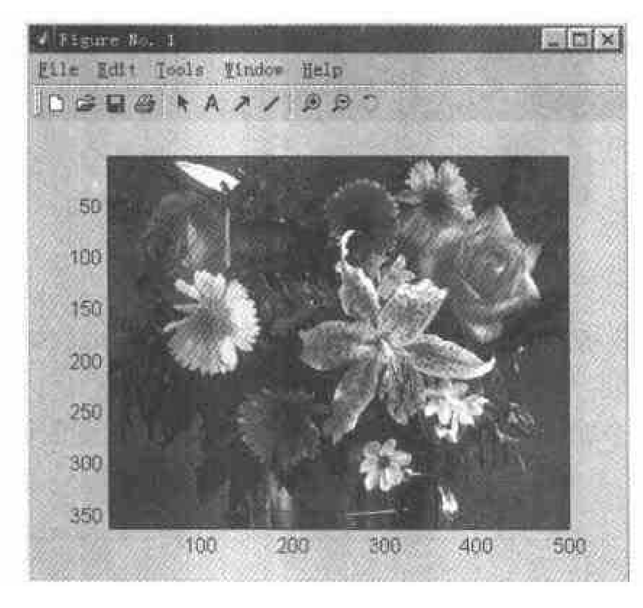


图 3-3 真彩图像

在上面的 RGB 图像中，要确定像素（12,9）的颜色，可以在命令行中键入：

```
RGB(12,9,:)
```

回车，得到：

```
ans(:,:,1) = 59
ans(:,:,2) = 55
ans(:,:,3) = 91
```



即像素（12,9）的RGB颜色为：59（红色）、55（绿色）、91（蓝色）。

如果 MATLAB 在一台不支持真彩显示的计算机上运行，则 MATLAB 用颜色近似和抖动来显示相近的真彩颜色。

3.2.4 二值图像

与灰度图像相同，二值图像只需要一个数据矩阵，每个像素只取两个灰度值。二值图像可以采用uint8或double类型存储，工具箱中以二值图像作为返回结果的函数都使用uint8类型。

图3-4是一幅二值图像的结构。

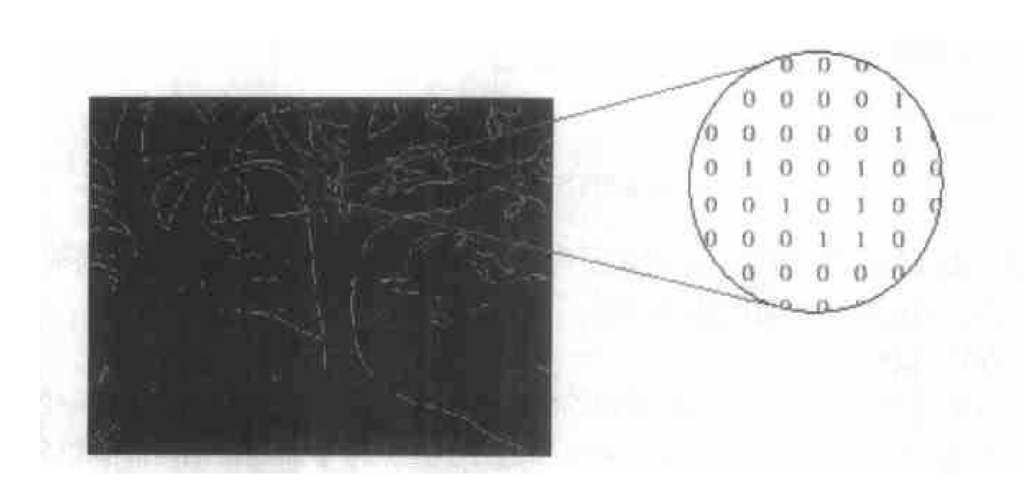


图3-4 二值图像的结构

3.2.5 图像序列

MATLAB 的图像处理工具箱还支持将多帧图像连接成图像序列。图像序列是一个四维的数组，图像帧的序号在图像的长、宽、颜色深度之后构成第四维。比如一个包含了5幅400×300（像素）真彩图像的序列，其大小为$400\times300\times3\times5$。



要将分散的图像合并成图像序列可以使用 MATLAB 的 cat 函数，前提是各图像的尺寸必须相同。如果是索引图像，颜色映射表也必须是一样的。

比如要将A1、A2、A3、A4、A5五幅图像合并成一个图像序列A，MATLAB命令为：

```
A=cat(4,A1,A2,A3,A4,A5)
```

也可以从图像序列中抽出一帧，比如代码：

```
FRM3=MULTI(:,:,:,3)
```

表示将序列MULTI中的第三帧提取出来赋给矩阵 ***FRM3***。

3.3 图像类型转换及示例

许多图像处理工作都对图像类型有特定的要求。比如要对一幅索引图像滤波，首先必须将它转换成真彩图像，否则结果是毫无意义的。

在 MATLAB 中，各种图像类型之间的转换关系如图 3-5 所示。

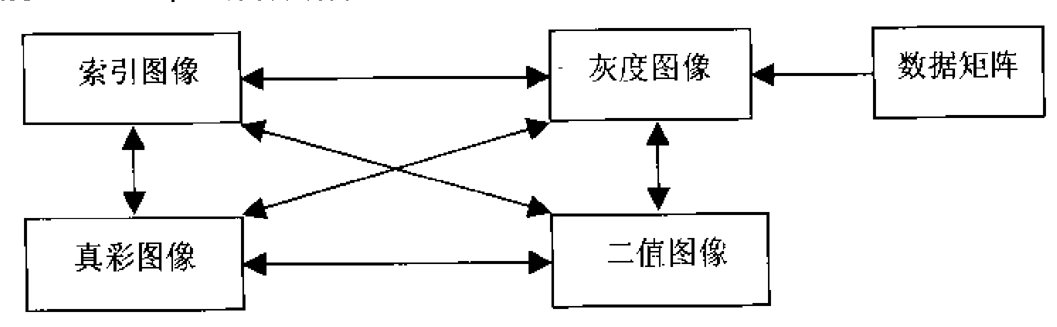


图 3-5 图像类型间的转换

MATLAB 的图像处理工具箱提供了许多图像类型转换函数，来实现上述的各种转换，下面就具体介绍这些函数的用法和功能。

- dither 函数

该函数通过颜色抖动（颜色抖动即改变边沿像素的颜色，使像素周围的颜色近似于原始图像的颜色，从而以空间分辨率来换取颜色分辨率）来增加输出图像的颜色分辨率，从而实现转换图像。

该函数的调用格式如下：

```
X = dither(RGB,map)
BW = dither(I)
```

X = dither(RGB,map)表示将真彩图像 RGB 按指定的颜色映射表 map 抖动成索引图像 X。

BW = dither(I)表示将灰度图像 I 抖动成二值图像 BW，

另外，输入图像可以是 double 或 uint8 类型，输出图像如果是二值图像或颜色种类不超过 256 的索引图像，则是 uint8 类型，否则为 double 型。

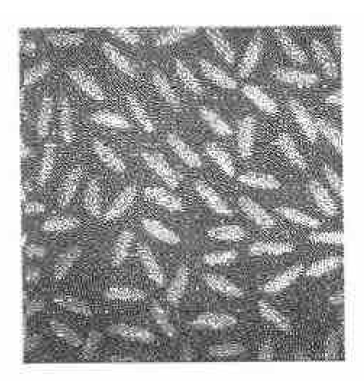



图 3-6 灰度图像转换成二值图像

例程 3-4 将灰度图像 rice.tif 转换为二值图像。

例程 3-4

```
I=imread('rice.tif');
BW=dither(I);
imshow(BW);
```

原图及转换后的结果如图 3-6 所示。

- gray2ind 函数

该函数的功能是将灰度图像转换成索引图像。其调用格式如下：

[X, map] = gray2ind(I,n)

[X, map] = gray2ind(I,n)表示按指定的灰度级数 n 将灰度图像 I 转换成索引图像 X，n 的默认值为 64。

另外，输入图像 I 可以是 double 类型、uint8 类型或 uint16 类型。如果颜色映射表的长度小于等于 256，则输出图像是 uint8 类型，否则为 double 类型。

例程 3-5 把一幅灰度图像转换为一幅索引图像。

例程 3-5

```
I=imread('rice.tif');
[X,map]=gray2ind(I,6);
imshow(X,map);
```

原图与转换后的图像的比较如图 3-7 所示。

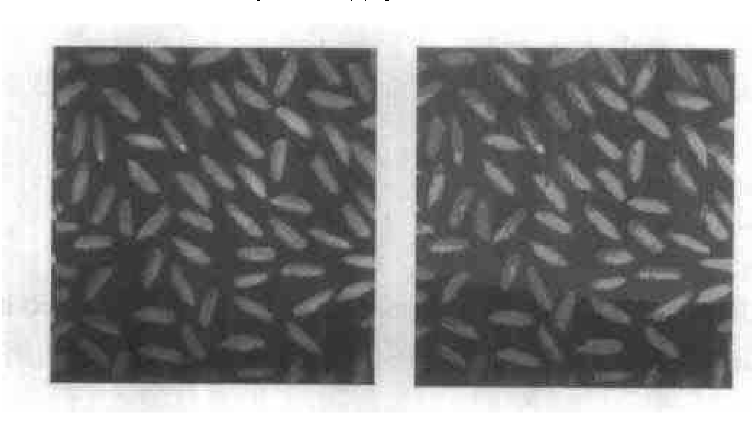


图 3-7　灰度图像转换成索引图像

- grayslice 函数

该函数的功能是通过设定阈值将灰度图像转换成索引图像。其调用格式如下：

X = grayslice(I,n)

X = grayslice(I,v)

X = grayslice(I,n) 将灰度图像 I 均匀量化为 n 个等级，然后转换为伪彩色图像 X。

X = grayslice(I,v)按指定的阈值矢量 v（其每一个元素都在 0 和 1 之间）对图像 I 的值域进行划分，而后转换成索引图像 X。

另外，输入图像 I 可以是 double 或 uint8 类型。如果阈值数量小于 256，则返回图像 X 的数据类型是 uint8，X 的值域为[0, n]或[0, length(v)]。否则，返回图像 X 为 double 类型，值域为[1, n+1]或[1, length(v)+1]。

例程3-6将一幅灰度图像转换成索引图像。

例程 3-6

```
I = imread('ngc4024m.tif');
X = grayslice(I,16);
imshow(I)
figure, imshow(X,hot(16))
```

原图与转换后的图像的比较结果见图 3-8。

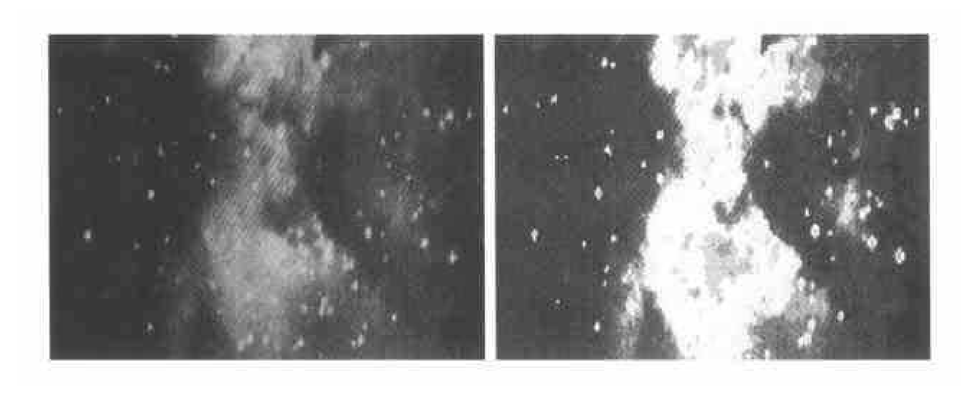


图 3-8　灰度图像转换成索引图像

- im2bw 函数

该函数的功能是通过设置亮度阈值将真彩、索引、灰度图像转换成二值图像。该函数的调用格式如下：

```
BW = im2bw(I,level)
BW = im2bw(X,map,level)
BW = im2bw(RGB,level)
```

BW = im2bw(I,level) 、BW = im2bw(X,map,level)和 BW = im2bw(RGB,level)分别表示将灰度图像、索引图像和真彩图像二值化为图像 BW。level 是归一化的阈值，取值在[0, 1]之间。

另外，输入图像可以是 double 或 uint8 类型，输出图像为 uint8 类型。

例程 3-7 将一幅索引图像转换成二值图像。

例程 3-7

```
load trees
BW = im2bw(X,map,0.4);
imshow(X,map)
figure, imshow(BW)
```

原图与转换后的图像的比较结果见图 3-9。







图 3-9 索引图像转换成二值图像

- ind2gray 函数

该函数的功能是将索引图像转换成灰度图像。其调用格式如下：

```
I = ind2gray(X,map)
```



输入图像可以是 double 或 uint8 类型，输出图像为 double 类型。

例程3-8 将一幅索引图像转换成了一幅灰度图像。

例程 3-8

```
load trees
I = ind2gray(X,map);
imshow(X,map)
figure,imshow(I)
```

原图与转换后的图像的比较结果如图 3-10 所示。



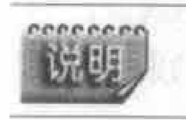


图 3-10 索引图像转换成灰度图像

- ind2rgb 函数

该函数的功能是将索引图像转换成真彩图像。其调用格式如下：

```
RGB = ind2rgb(X,map)
```



输入图像 X 可以是 double 或 uint8 类型，输出图像 RGB 为 double 类型。

- mat2gray 函数

该函数的功能是将一个数据矩阵转换成一幅灰度图像。其调用格式如下：

```
I = mat2gray(A,[amin amax])
I = mat2gray(A)
```

I = mat2gray(A,[amin amax])表示按指定的取值区间[amin amax]将数据矩阵 A 转化为灰度图像 I，amin 对应灰度 0（最暗），amax 对应 1（最亮）。如果不指定区间[amin amax]，则 MATLAB 自动将 A 阵中的最小元设为 amin，最大元设为 amax。

另外，输入矩阵 A 与输出图像 I 都为 double 类型。

例程 3-9 Sobel 算子对图像滤波，将滤波得到的数据矩阵转换为灰度图像。

例程 3-9

```
I = imread('rice.tif');
J = filter2(fspecial('sobel'),I);
K = mat2gray(J);
imshow(I)
figure, imshow(K)
```

原图与转换后的图像的比较结果见图 3-11。

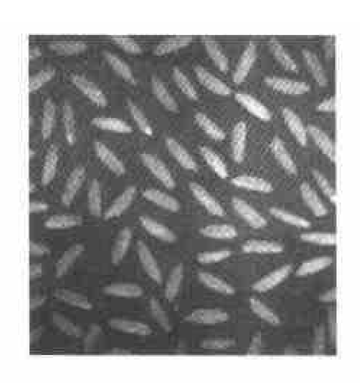



图 3-11　将数据矩阵转换成灰度图像

- rgb2gray 函数

该函数的功能有两个：第一个功能是将一幅真彩图像转换成一幅灰度图像；第二个功能是把一幅索引图像的彩色颜色映射表（即颜色映射表）转换为灰度颜色映射表。其调用格式如下：

```
I = rgb2gray(RGB)
newmap = rgb2gray(map)
```

I = rgb2gray(RGB)表示将真彩图像 RGB 转换成灰度图像 I。

newmap = rgb2gray(map)表示将彩色颜色映射表 map 转化成灰度颜色映射表。

需要注意的是，如果输入的是真彩图像，则可以是 uint8 或 double 类型，输出图像 I 与输入图像类型相同。如果输入的是颜色映射表，则输入、输出都是 double 类型。

例程 3-10 将一幅彩色索引图像转换为一幅颜色映射表为灰度颜色映射表的索引图像。

例程 3-10

```
[X,map]=imread('canoe.tif');
newmap=rgb2gray(map);
imshow(X,newmap);
```

- rgb2ind 函数

该函数的功能是将索引图像转换成真彩图像。其调用格式如下：

RGB = ind2rgb(X,map)

X 和 map 分别代表原输入索引图像的数据矩阵和颜色映射表。

另外，输入图像 X 可以是 double 或 uint8 类型，输出图像 RGB 为 double 类型。

3.4 MATLAB 中的 8 位和 16 位图像

MATLAB 中最基本的数据结构是数组。在 MATLAB 中，大多数图像用二维数组（矩阵）存储，矩阵中的一个元素对应于所要显示图像的一个像素。例如，一个由 100 行和 200 列的不同颜色的点组成的图像可以用一个 100×200 的矩阵来存储。也有一些图像，如 RGB 图像，需要用三维数组来存储，第一维表示红色像素的深度值，第二维表示绿色像素的深度值，第三维表示蓝色像素的深度值。

这意味着 MATLAB 强大的矩阵运算功能完全可以应用于图像，那些适用于矩阵运算的语法对 MATLAB 中的数字图像同样适用。

在默认的情况下，MATLAB 将图像中的数据存储为双精度类型（double），即 64 位的浮点数。这种存储方法的优点在于使用中不需要数据类型转换，因为几乎所有的 MATLAB 及其工具箱函数都可使用 double 作为参数类型。然而对于图像存储来说，用 64 位来表示图像数据会导致巨大的存储量，所以 MATLAB 还支持图像数据的另一种类型——无符号整型（uint8），即图像矩阵中的每个数据占用一个字节。MATLAB 及工具箱中的大多数操作及函数（比如最基本的矩阵相加）都不支持 uint8 类型。uint8 的优势仅仅在于节省存储空间，在涉及运算时要将其转换成 double 型。

在处理图像时，MATLAB 通常使用 64 位的双精度浮点类型。但是，为了减少内存需求，提高系统运行效率，MATLAB 还提供了两个重要的数字类型 uint8 和 uint16，用以支持 8 位和 16 位的无符号整数类型。在 MATLAB 中，数据矩阵中包含 uint8 数字类型的图像称之为 8 位图；同理，数据矩阵中包含 uint16 数字类型的图像称为 16 位图。

利用 MATLAB 提供的 image 函数，可以直接显示 8 位图像或 16 位图像，而不必将其转换为双精度浮点类型。然而，当图像是 uint8 或 uint16 类型时，MATLAB 对矩阵值的解释有所不同，这主要依赖于图像的具体类型。

3.4.1 8 位和 16 位索引图像

如果图像数据矩阵的类型是 uint8 或 uint16，则其值在作用于颜色映射表的索引之前，必须进行值为 1 的偏移，即值 0 指向矩阵 map 的第一行，值 1 指向第二行，依此类推。

并且因为 image 函数自动提供了这种偏移，所以不管图像数据矩阵是双精度浮点类型，还是 uint8 或 uint16，显示的方法均相同。

但是，当用户在进行类型转换时，由于偏移量的存在，必须将 uint8 或 uint16 的数据加 1，然后才能将其转换为双精度浮点类型。例如：

```
X64=double(X8)+1;
```

或

```
X64=double(X16)+1;
```

相反，若将双精度浮点类型转换为 uint8 或 uint16 类型的数据时，在转换之前，必须将其减去 1。例如：

```
X8=uint8(X64.1);
```

或

```
X16=uint16(X64.1);
```

3.4.2　8 位和 16 位灰度图像

在 MATLAB 中，双精度浮点类型的图像数组中的数值的取值范围通常为[0，1]，而 8 位和 16 位无符号整数类型的图像的取值范围分别为[0，255]和[0，65535]。

例程 3-11

```
I=imread('rice.tif');
Imagesc(I,[0 255]);
colormap(gray);
```

在 MATLAB 中，将一个灰度图从双精度的浮点类型转换为 uint16 的 16 位无符号整数类型，必须首先将其乘以 65535。例如：

```
I16=uint16(round(I64*65535));
```

于此相反，将一个灰度图从 uint16 的 16 位无符号整数类型转换为双精度的浮点类型时，必须首先除以 65535。例如：

```
I64=double(I16)/65535;
```

3.4.3　8 位和 16 位 RGB 图像

8 位 RGB 图像的颜色数据是[0,255]之间的整数，而不是[0，1]之间的浮点值。所以，在 8 位 RGB 图像中，颜色值为（255,255,255）的像素显示为白色。不管 RGB 图像是何种类型，MATLAB 都通过以下的代码来显示，即：

```
image(RGB);
```

将 RGB 图像从双精度的浮点类型转换为 uint8 无符号整数类型时，首先要乘以 255。即：

```
RGB8=uint8(round(RGB64*255));
```

相反，如果将 uint8 无符号整数类型的 RGB 图像转换为双精度的浮点类型时，首先要除以 255。即：

```
RGB64=double(RGB8)/255;
```

另外，如果将 RGB 图像从双精度的浮点类型转换为 uint16 无符号整数类型时，必须乘以 65535，即：

```
RGB16=uint16(round(RGB64*65535));
```

同样，如果将 uint16 无符号整数类型的 RGB 图像转换为双精度浮点类型时，首先要除以 65535。即：

```
RGB64=double(RGB16)/65535;
```

3.5 图像文件的读写和查询方法

在对一幅图像文件进行处理时，先抛开具体的处理过程不说，我们首先要熟悉以下的几个操作：

- 查询该图像的相关信息。
- 读取该图像的相关数据。
- 将处理过的图像数据保存到另一个图像文件中。

针对上述的几个操作，MATLAB 都给用户提供了相关的函数。下面就详细地介绍这几个相关的函数。

3.5.1 图像文件信息的查询

MATLAB 中，用于图像文件信息查询的是 imfinfo 函数。其格式如下：

```
info = imfinfo('文件名',文件格式)
info = imfinfo('文件名')
```

由该函数获取的信息依赖于文件类型的不同而不同，但至少应包含以下内容：

- '文件名'（文件名）
- FileModDate（文件最后一次的修改时间）
- FileSize（文件的大小，单位为字节）
- Format（文件格式）
- FormatVersion（文件格式的版本号）
- Width（图像的宽度，单位为像素）
- Height（图像的高度，单位为像素）
- BitDepth（每个像素的位数）
- ColorType（图像类型，即是 RGB 图像、灰度图像还是索引图像）

例如，在 MATLAB 的命令行中输入下面的代码，以查询文件 rice.tif 的信息。

```
info=imfinfo('rice.tif')
```

执行后，得到的结果如下：

```
info =
'文件名': 'rice.tif'
          FileModDate: '25.Oct.1996 22:11:58'
          FileSize: 65966
            Format: 'tif'
```

```
FormatVersion: []
Width: 256
    Height: 256
BitDepth: 8
ColorType: 'grayscale'
FormatSignature: [73 73 42 0]
ByteOrder: 'little.endian'
NewSubfileType: 0
BitsPerSample: 8
Compression: 'Uncompressed'
PhotometricInterpretation: 'BlackIsZero'
StripOffsets: [8x1 double]
SamplesPerPixel: 1
RowsPerStrip: 32
StripByteCounts: [8x1 double]
XResolution: 72
YResolution: 72
ResolutionUnit: 'Inch'
Colormap: []
PlanarConfiguration: 'Chunky'
TileWidth: []
  TileLength: []
TileOffsets: []
TileByteCounts: []
Orientation: 1
FillOrder: 1
GrayResponseUnit: 0.0100
MaxSampleValue: 255
  MinSampleValue: 0
Thresholding: 1
ImageDescription: [1x166 char]
```

3.5.2 图像文件的读取

MATLAB 中利用函数 imread 来实现图像文件的读取操作。其格式如下：

```
A = imread('文件名',文件格式)等价于 A = imread('文件名.文件格式')
[X, map] = imread('文件名',文件格式)
[…] = imread('文件名')
[…] = imread(…,idx) (只适用于*.tif 格式)
[…] = imread(…,ref) （只适用于*.HDF 格式）
```

[…] = imread(…,idx) 用于读取多帧 TIFF 文件中的一帧。

[…] = imread(…,ref) 用于读取多帧 HDF 文件中的一帧。

通常情况下，我们通过 imread 函数读取的大多数图像都是 8 位的。当把这些图像加载到内存中时，MATLAB 就将其存储在类 uint8 中。

此外，MATLAB 还支持 16 位的 PNG 和 TIFF 图像。当用户读取这类图像时，MATLAB

就将其存储在类 uint16 中。而对于索引图像来说，即是图像矩阵的本身为 uint8 或 uint16，imread 函数仍将颜色映射表读取并存储在一个双精度浮点类型的矩阵中。

3.5.3　图像文件的保存

MATLAB 中利用函数 imwrite 来实现图像文件的保存操作。其格式如下：

```
imwrite(A,'文件名',文件格式)
imwrite(X,map,'文件名',文件格式)
imwrite(…,'文件名')
imwrite(…,Parameter,Value,…)
```

文件格式可以取以下的值：'bmp'、'hdf'、'jpg'、'pcx'、'png'、'tif'和'xwd'。

imwrite(…,'文件名')中的文件名必须带合法的扩展名。

imwrite(…,Parameter,Value,…)可以让用户控制 HDF、JPEG、TIFF 三种图像文件的输出特性。

对于 HDF 文件，参数说明见表 3-1。

表 3-1　对于 HDF 文件有效的参数的说明

Parameter	可 选 值	
'Compression'	'none'、'rle'、'jpeg'，其中，'rle'只对灰度图像和索引图像有效；'jpeg'值对灰度图像和 RGB 图像有效	'rle'
'Quality'	取值为 0 到 100，只有 Compression 为'jpeg'时才有效，且其值越大，压缩质量越高，但文件也越大	75
'WriteMode'	'overwrite'或'append'	'overwrite'

对于 JPEG 文件，参数说明见表 3-2。

表 3-2　对于 jpeg 文件有效的参数的说明

Parameter	可 选 值	
'Quality'	取值为 0 到 100，值越大，压缩质量越高，但文件也越大	75

对于 TIFF 文件，参数说明见表 3-3。

表 3-3　对于 TIFF 文件有效的参数的说明

Parameter	可 选 值	默 认 值
'Compression'	'none'、'packbits'、'ccitt'，其中'ccitt'只对二值图像有效	对于二值图像是'none'，对于其他图像是'packbits'
'Description'	任意字符串，填充由函数 imfinfo 返回的 ImageDescription 域	空
'Resolution'	一个标量值，用来定义输出文件在 x 和 y 方向的分辨率	72

例程 3-12 将一幅.tif 格式的图像保存为一幅.hdf 格式的图像。

例程 3-12

```
[X,map]=imread('canoe.tif');
imwrite(X,map,'canoe.hdf','Compression','none', 'WriteMode','append')
```

另外，需要注意的是，当利用 imwrite 函数保存图像时，MATLAB 默认的保存方式是将其简化为 uint8 的数据类型。与读取图像文件类型类似，MATLAB 在文件保存时还支持 16 位的 PNG 和 TIFF 图像。所以，当用户保存这类文件时，MATLAB 就将其存储在 uint16 中。

3.6 图像对象属性详解

MATLAB 中，可以调用 image 函数和 imagesc 函数来创建图像对象。与线条对象、片块对象、表面对象以及文本对象类似，图像对象也是坐标轴对象的子对象。另外，与所有的句柄图形对象一样，图像对象也包含许多用户可以设置的以调整其屏幕显示外观的属性。其中最重要的属性包括 CData 属性、CDataMapping 属性、XData 属性和 YData 属性等。

3.6.1 图像对象的 CData 属性

图像对象的属性 CData 包含一个数据阵列。下面的代码中，h 表示由 image 函数创建的图像对象的句柄，X 和 Y 代表同一个矩阵。

```
H=image(X);
colormap(map);
Y=get(h,'CData')
```

其中，属性 CData 数据阵列的维数决定了 MATLAB 是使用颜色映射表还是使用 RGB 真彩的方式显示该图像。如果 CData 属性的数据阵列是二维的，则图像显示为索引图像或灰度图像。但不管是何种方式，MATLAB 均使用颜色映射表。如果 CData 属性的数据阵列是三维的，形如 $m\times n\times 3$，那么 MATLAB 就将其显示为真彩图像。

3.6.2 图像对象的 CDataMapping 属性

图像对象的 CDataMapping 属性决定了图像是索引图像还是灰度图像。如果将该属性设置为“direct”，则该图像显示为索引图像，在这种情况下，CData 属性中的数据阵列直接作为当前图形窗口的颜色映射表中的颜色索引。在 MATLAB 中，只要我们在调用 image 函数时只使用了一个参数，则系统自动将 CDataMapping 属性的值设置为“direct”。

例如，我们在命令行中输入下面的代码：

```
h=image(X);
colormap(map);
```

```
get(h,'CDataMapping');
```

回车执行后，得到下面的结果：

```
ans=
direct
```

另外，如果将 CDataMapping 属性的值设置为“scaled”，则 MATLAB 将该图像对象显示为灰度图像。在这种情况下，MATLAB 将 CData 属性的数据线性缩放，生成颜色映射表的索引，其中缩放因子由坐标轴对象的 CLim 属性确定。

在 MATLAB 中，只要利用 imagesc 函数创建图像对象，系统自动将 CDataMapping 属性的值设置为“scaled”。并且同时调整其父坐标轴对象的 CLim 属性。

例如，在命令行中输入下面的代码：

```
h=imagesc(I,[0 1]);
colormap(map);
get(h,'CDataMapping')
```

回车执行后，得到下面的结果：

```
ans=
scaled
```

输入下面的代码：

```
get(gca,'CLim')
```

执行后，得到下面的结果：

```
ans=
[0 1]
```

3.6.3 图像对象的 XData 和 YData 属性

图像对象的 XData 和 YData 属性决定了图像的坐标系统，对于一个数据矩阵为 $m \times n$ 的图像对象来说，其 XData 属性的默认值是[1 n]，YData 属性的默认值是[1 m]。这样的设置意义如下：

- 图像数据矩阵的左边的列具有一个值为 1 的 x 坐标。
- 图像数据矩阵的右边的列具有一个值为 n 的 x 坐标。
- 图像数据矩阵的上边的行具有一个值为 1 的 y 坐标。
- 图像数据矩阵的下边的行具有一个值为 m 的 y 坐标

例程 3-13 在默认的坐标系中创建一个矩阵图像对象。

例程 3-13

```
X=[1 2 3 4;5 6 7 8;1 2 3 4];
h=image(X);
colormap(map)
xlabel('x 轴');
ylabel('y 轴');
```

生成如图 3-12 所示的矩阵图像对象。

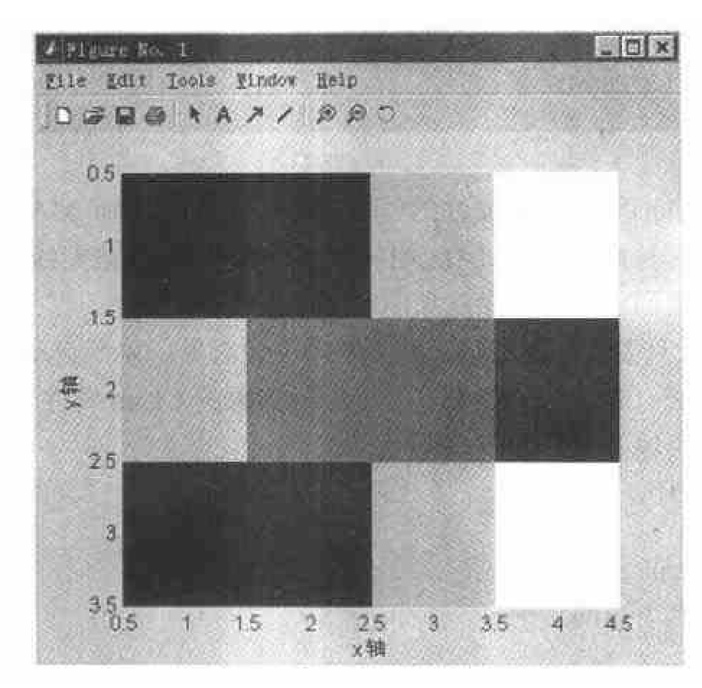


图 3-12　默认坐标系下的图像显示

这样生成的图像对象的 XData 和 YData 属性具有默认的值。在命令行中输入下面的代码：

```
get(h,'XData')
```

执行后，得到如下结果：

```
ans=
1    4
```

同样，代码：

```
get(h,'YData')
```

执行的结果如下：

```
ans=
1    3
```

另外，用户还可以定义自己的坐标系来覆盖系统的默认的设置。

例程 3-14 在自定义的坐标系中创建一个矩阵图像对象。

例程 3-14

```
X=[1 2 3 4; 5 6 7 8;1 2 3 4];
image(X,'XData',[1 2],'YData',[2 4]);
colormap(map);
xlabel('x 轴');
ylabel('y 轴');
```

定义一个新的坐标系，其结果如图 3-13 所示。

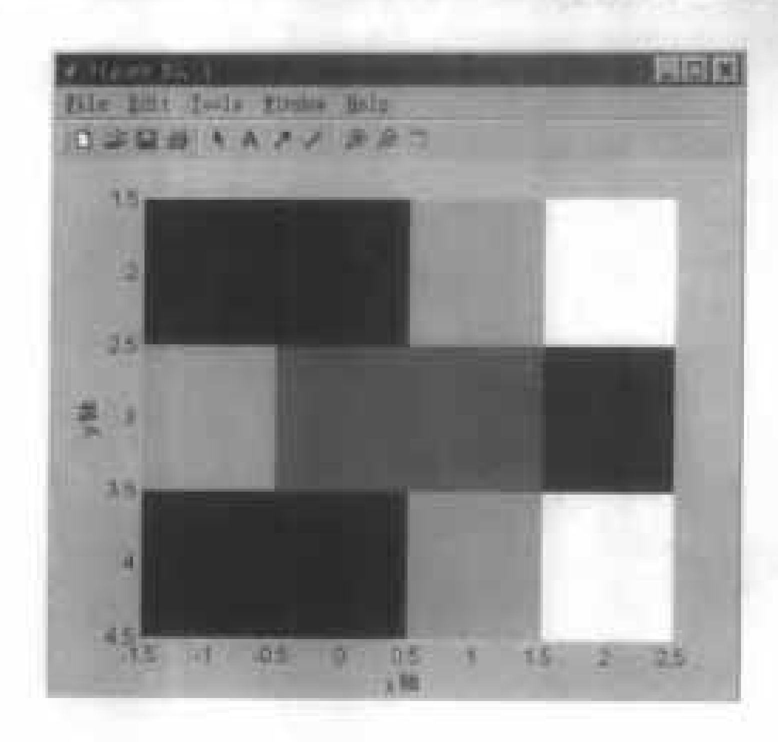


图 3-13 自定义坐标系下的图像显示

第4章 MATLAB 图像显示技术

MATLAB 颜色模型 MATLAB 的图像处理工具箱提供了多种图像显示和颜色处理技术。本章将向读者具体介绍 MATLAB 中用于图像显示及用于颜色模型转换的各个函数的具体用法。

本章内容包括：

- 标准的图像显示技术
- 特殊的图像显示技术

4.1 标准的图像显示技术

由第 3 章已经知道，我们可以调用 image 函数和 imagesc 函数来显示一幅图像。在 MATLAB 的图像处理工具箱中，还提供了一个应用很广泛的图像显示函数，即 imshow 函数。与 image 函数和 imagesc 函数类似，imshow 函数也创建句柄图形图像对象。此外，imshow 函数也可以自动设置各种句柄图形属性和图像特征，以优化显示效果。

4.1.1 imshow 函数

当用户调用 imshow 函数显示一幅图像时，该函数将自动设置图像窗口、坐标轴和图像属性。这些自动设置的属性包括图像对象的 CData 属性和 CDataMapping 属性、坐标轴对象的 CLim 属性、图像窗口对象的 Colormap 属性。

另外，根据用户使用参数的不同，imshow 函数在调用时除了完成前面提到的属性设置外，还可以完成下面的操作：

1）设置其他的图形窗口对象的属性和坐标轴对象的属性以优化显示效果。如，可以设置隐藏坐标轴及其标示等。

2）包含和隐藏图像边框。

3）调用 truesize 函数以显示没有彩色渐变效果的图像。

imshow 函数的调用格式如下：

```
imshow(I,n)
imshow(I,[low high])
imshow(BW)
imshow(X,map)
imshow(RGB)
imshow(…,display_option)
imshow(x,y,A,…)
imshow filename
```

每一种格式的具体用途将在下面详细介绍

4.1.2 显示灰度图像

调用 imshow 函数显示灰度图像的格式如下：

imshow(I,n)

imshow(I,[low high])

I 代表所显示的灰度图像的数据矩阵。

n 为整数，代表所要显示图像的灰度等级数，默认值为 256。

[low high]为图像数据的值域。需要注意的是，在很多情况下，经过处理的图像数据的值域都会发生变化。比如对一幅 double 型的灰度图像滤波后，图像数据的值域已不在[0,1]中了，如果还用前面的显示方法，则得不到正确的结果。如果清楚地知道数据的值域[low high]，可以调用 imshow(I,[low high])。否则可以用空矢量为参数，即 imshow(I,[ ])。

例程 4-1 在一定灰度范围内显示灰度图像。

例程 4-1

```
I=imread('rice.tif');
imshow(I,[100 200])
```

可以得到如图 4-1 所示的结果。

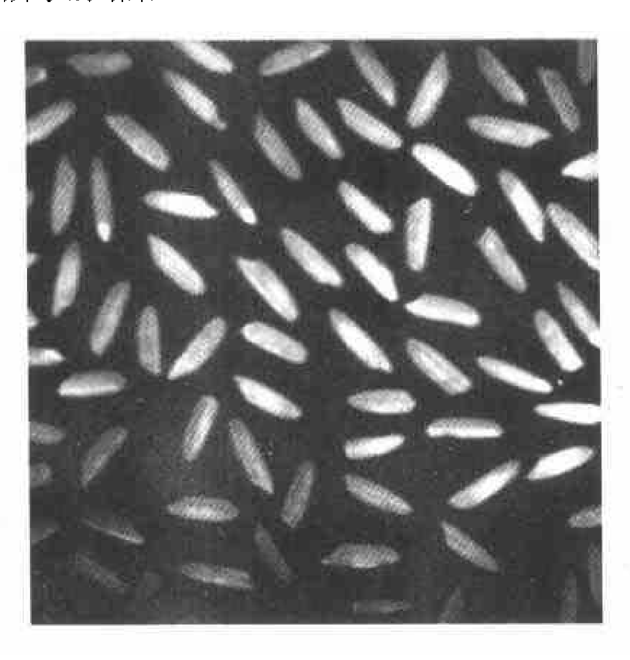


图 4-1 定义了灰度范围的灰度图像

例程 4-2 按照指定的灰度等级显示灰度图像。

例程 4-2

```
I=imread('rice.tif');
imshow(I,20)
```



可以得到如图 4-2 所示的结果。

4.1.3　显示二值图像

调用 imshow 函数显示二值图像的格式如下：

```
imshow(BW)
```



如果该二值图像矩阵属于 double 类，则 imshow 函数将其视为灰度图来对待，同时将 CDataMapping 的属性值设置为 scaled，CLim 的属性值设置为[0 1]；Colormap 颜色映射表的属性值设置为灰度颜色映射表。所以，在这种情况下，图像数据矩阵中值 0 所对应的像素显示为黑色，而值 1 所对应的像素显示为白色。

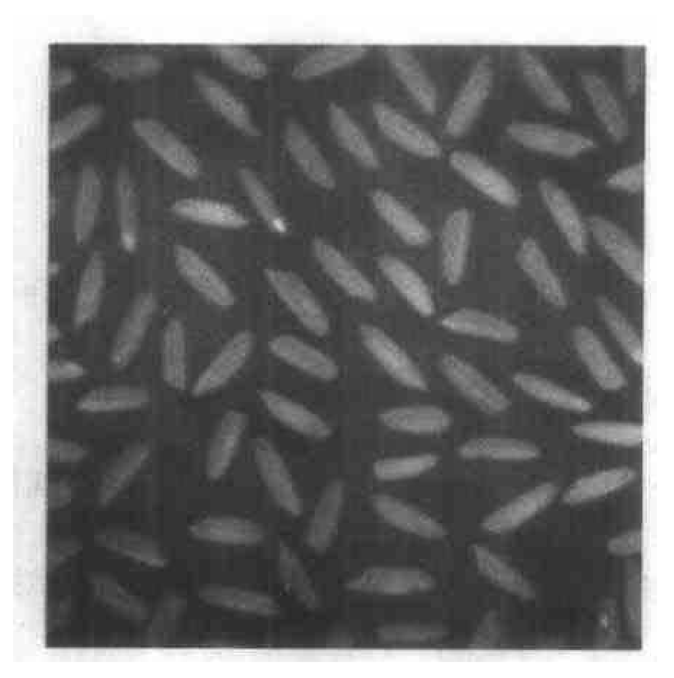


图 4-2　定义了灰度等级的灰度图像

例程 4-3 显示一幅二值图像。

例程 4-3

```
BW=imread('circles.tif');
imshow(BW)
```

得到如图 4-3 所示的结果。

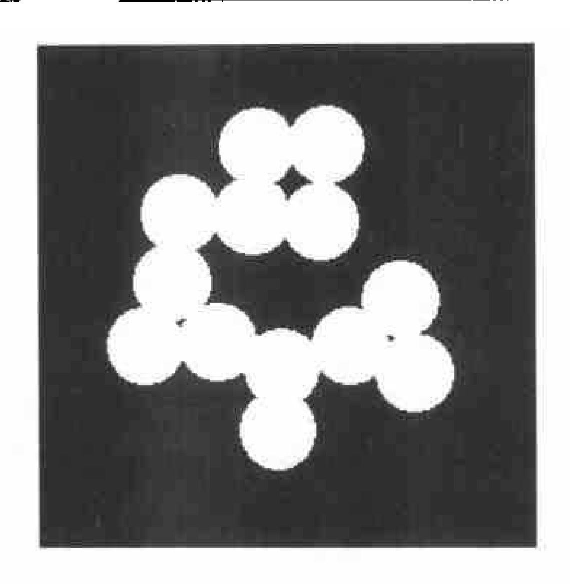


图 4-3 二值图像

4.1.4 显示索引图像

调用 imshow 函数显示索引图像的格式如下：

```
mshow(X,map)
```

其中，X 代表索引图像的数据矩阵，map 为颜色映射表。

另外，显示索引图像时，imshow 函数将同时设置下面的一些用来控制显示颜色的句柄图形的属性：

- 将图像的 CData 属性值设置为 X 矩阵中的数据。
- 将图像的 CDataMapping 属性值设置为 direct。
- 使坐标轴对象的属性失效，因为图像的 CDataMapping 属性值已经被设置为了 direct。
- 将图形窗口对象的 Colormap 属性值设置为 map 矩阵中的数据。

4.1.5 显示真彩图像

调用 imshow 函数显示真彩图像的格式如下：

```
imshow(RGB)
```

其中 RGB 为代表该真彩图像的 $m \times n \times 3$ 的数据阵列。

4.1.6 显示图形文件中的图像

前面介绍的几种用 imshow 函数显示图像的方法中，均调用了所显示图像的对象数据。如果所显示图像被保存在可以通过 imread 函数读取的图像文件中时，imshow 函数可以通过下面的格式直接将其显示出来：

```
imshow 文件名;
```



该文件名必须带有合法的扩展名（即指明文件格式），且该图像文件必须保存在当前目录下，或在 MATLAB 默认的目录下。

显示一幅在当前目录下的 .jpg 格式的图像：

```
imshow  pic005.jpg
```

显示的结果如图 4-4 所示。

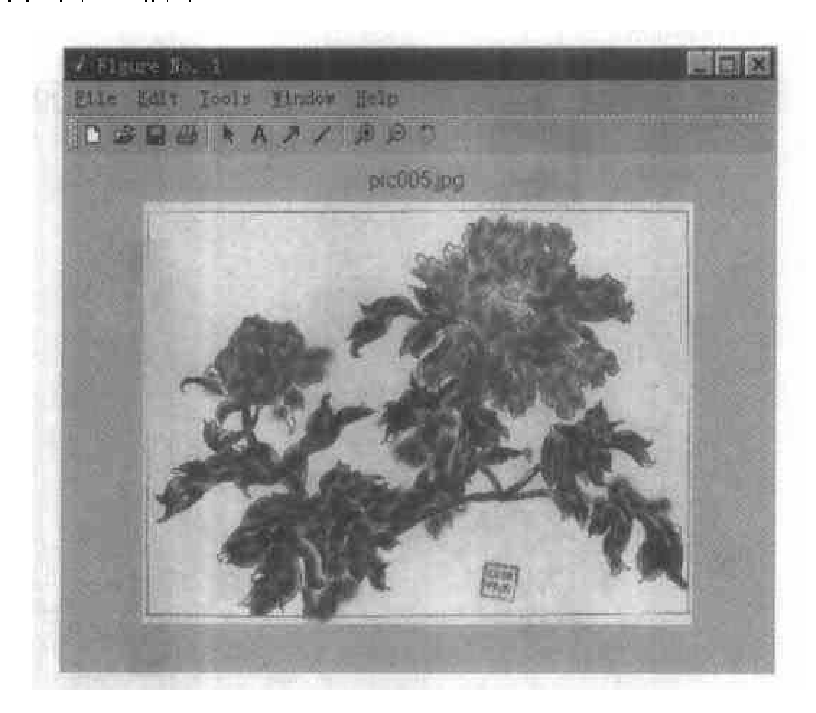


图 4-4　显示一幅图像文件中的图像

4.2　特殊图像显示技术

除了前面已经介绍的 imshow 函数以外，MATLAB 的图像处理工具箱还提供了一些可以实现特殊显示功能的函数。下面就具体介绍这些函数的用法和功能。

4.2.1　添加颜色条

在 MATLAB 的图像显示中，可以利用 colorbar 函数将颜色条添加到坐标轴对象中。添加进来的颜色条用来指示图像中不同颜色所对应的数据值。

colorbar 函数的语法格式如下：

```
colorbar('vert')
colorbar('horiz')
colorbar(h)
colorbar
h = colorbar(…)
```

colorbar('vert')、colorbar('horiz')指定了颜色条的显示方式为垂直或水平。默认值为垂

直('vert')；

colorbar(h)将颜色条放在指定的坐标轴 h 上，h 为句柄。

h = colorbar(…)返回颜色条坐标轴的句柄。

添加了颜色条后的图像显示：

```
imshow pic005.jpg;
colorbar
```

得到如图 4-5 所示的结果。

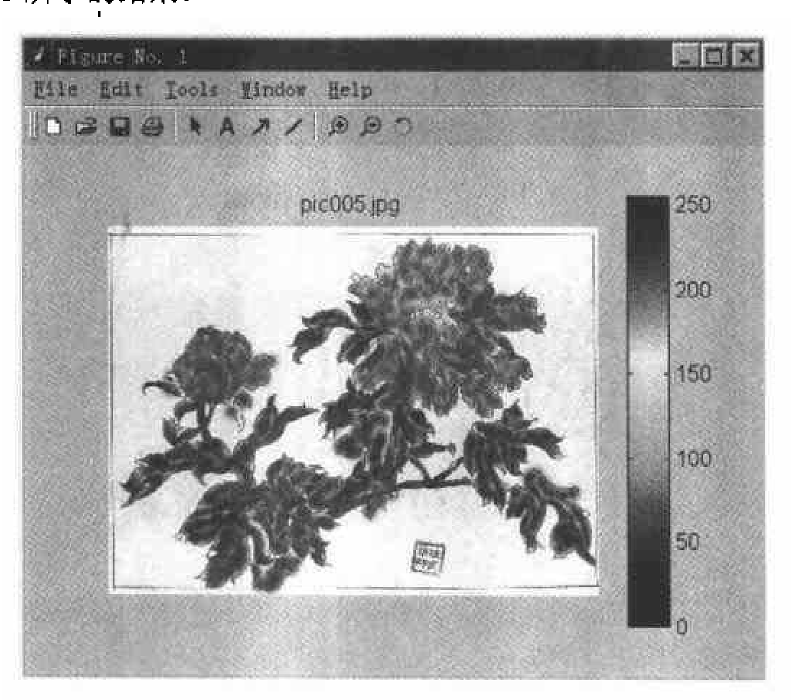


图 4-5　添加颜色条后的图像显示

4.2.2　显示多帧图像阵列

在 MATLAB 中，在多帧图像阵列中查看图像，有下面几种方式：

- 独立显示每一帧，调用imshow函数。
- 同时显示所有的帧，调用montage函数。
- 将多帧图像阵列转换为电影动画，调用immovie函数。

下面将具体介绍这几个函数的用法。

1. 单帧显示

下面通过例程 4-4 来说明图像的单帧显示。

例程 4-4

```
%将两幅灰度图像合并成一个具有两帧的图像阵列，然后再调用 imshow 函数来显示第
%一帧图像
A1=imread('rice.tif');
A2=imread('testpat1.tif');
```

```
A=cat(3,A1,A2);          %用 cat 函数实现矩阵的合并
imshow(A(:,:,1))         %1 代表第一帧
```

得到如图 4-6 所示的结果。

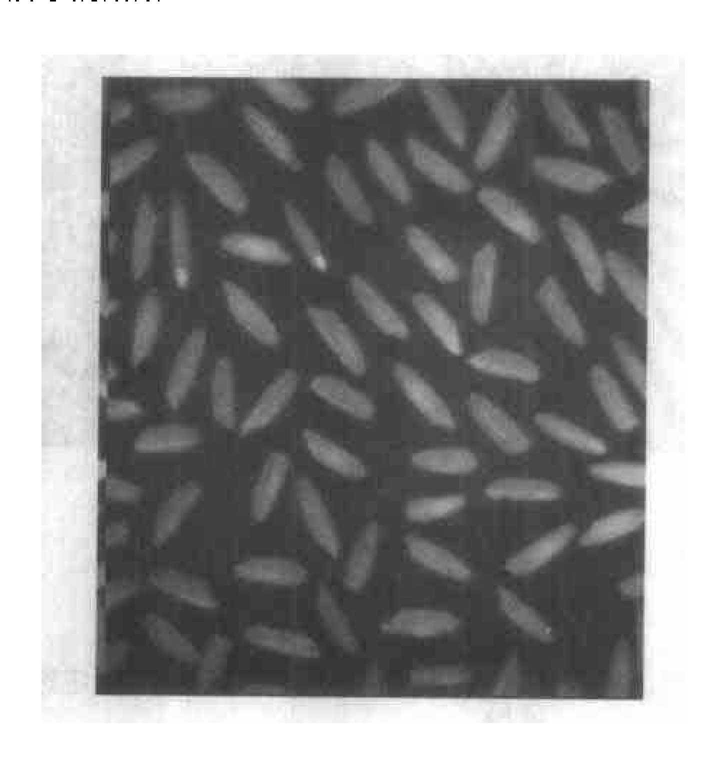


图 4-6　图像的单帧显示

2．多帧显示

在 MATLAB 中，要同时显示多帧图像阵列，需要调用 montage 函数。其格式具体如下：

```
montage(I)
montage(X,map)
montage(RGB)
h = montage(…)
```

例程 4-5 图像的多帧显示。

例程 4-5

```
A1=imread('rice.tif');
A2=imread('testpat1.tif');
A=cat(2,A1,A2);
montage(A)
```

得到如图 4-7 所示的结果。

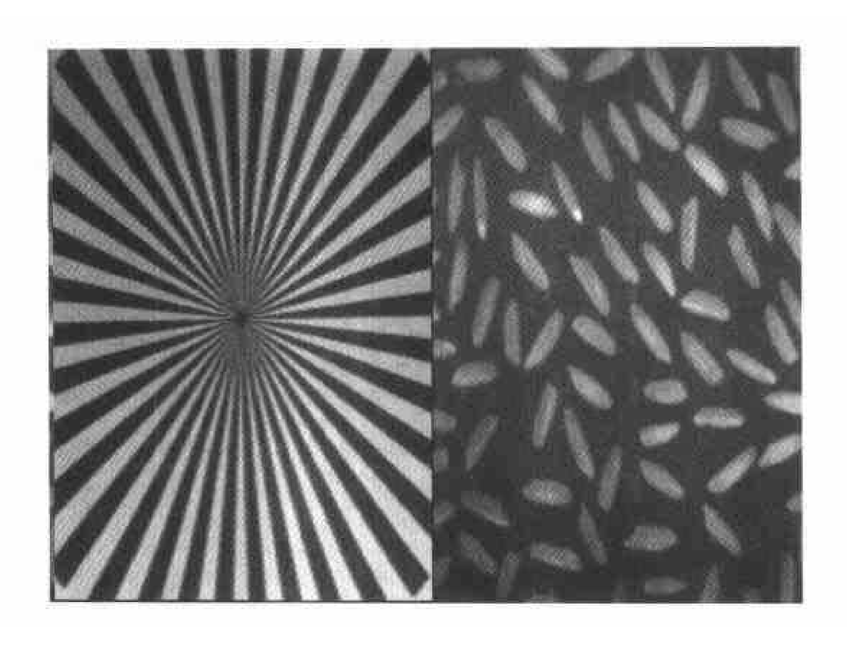


图 4-7　图像的多帧显示

3．动画显示

利用 immovie 函数，可以从多帧图像阵列中创建 MATLAB 电影动画。其调用格式如下：

```
mov=immovie(D,map)
```



该函数只能用于索引图像。所以，如果用户希望将其他类型的图像阵列转换为电影动画，必须首先将其类型转换为索引类型。而且函数中所带的参数 D 为一个 $m\times n\times 1\times k$ 的 4 维数组，k 代表图像阵列中所包含的帧的数目。

例程4-6中，两幅图像都是灰度图像，所以先利用gray2ind函数将它们转换为索引图像。然后还要注意利用已经得到的图像数据矩阵来求数组D。

例程 4-6

```
A1=imread('rice.tif');
A2=imread('testpat1.tif')
[B1,map]=gray2ind(A1,256);
[B2,map]=gray2ind(A2,256);      %将两幅灰度图像转换为两幅索引图像
A=cat(3,B1,B2,B1,B2,B1,B2,B1,B2);   %实现矩阵的合并
s=size(A);
for i=1:s(1,1)              %该循环用来求代表多帧图像阵列的 4 维矩阵
    for j=1:s(1,2)
        for k=1:s(1,3)
            D(i,j,1,k)=A(i,j,k);
        end
```

```
    end
end
mov=immovie(D,map);
```

4.2.3 图像上的区域缩放

MATLAB 中利用 zoom 命令来实现图像上任一区域的缩放。其调用格式如下：

```
zoom on
zoom off
```



zoom on 命令使用户可以用鼠标缩放图像。用户可以用鼠标选择一个图像点或拉出一个矩形框，单击左键放大图像，单击右键缩小图像。zoom off 命令关闭缩放功能。

例程 4-7 用 zoom 函数实现区域的缩放。

例程 4-7

```
imshow pic007.jpg;
zoom on
```

在打开的图像中按住并拖拉鼠标左键选择任意区域，再松开鼠标左键，则原图和放大后的图像分别如图 4-8 和图 4-9 所示。



图 4-8 原图像

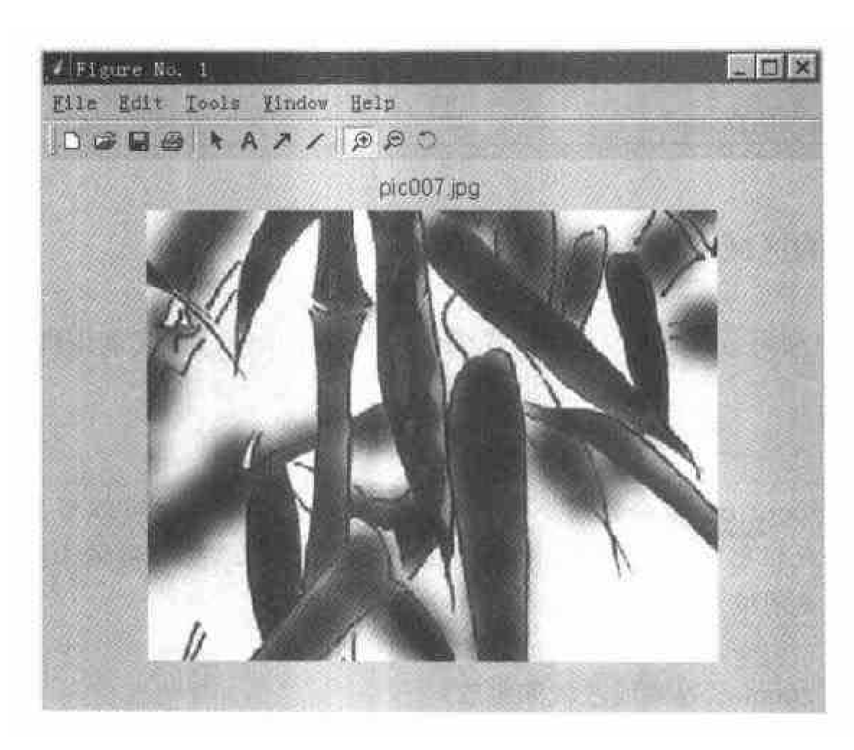


图 4-9　放大后的图像（局部）

4.2.4　纹理映射

在 MATLAB 中，纹理映射是一种将二维图像映射到三维图形表面的技术。这种技术通过转换颜色数据使二维图像与三维表面图形保持一致。

MATLAB 的图像处理工具箱中提供了一个专门的函数，即 warp 函数，将图像作为纹理进行映射，使该图像显示在一个特定的三维空间中。下面具体介绍该函数的用法。

warp 函数的调用格式如下：

```
warp(X,map)
warp(I,n)
warp(RGB)
warp(z,…)
warp(x,y,z,…)
h = warp(…)
```

warp(X,map)、warp(I,n)、warp(RGB)分别表示将索引色图像、灰度图像和真彩色图像映射到矩形平面区域上显示。由于矩形平面区域本身就是一个二维图形区域，所以调用这三种格式来显示图像与直接调用 imshow 函数的显示结果是一致的。

warp(z,…)表示将图像映射到曲面 Z 上。

warp(x,y,z,…)表示将图像映射到曲面（X,Y,Z）上。

h = warp(…)表示返回纹理映射后的图形句柄。

还有一点需要说明的是，MATLAB 中的纹理映射是利用双线性渐变算法来实现图像的映射的。

例程 4-8 将一幅真彩图像映射到球面上。

例程 4-8

```
RGB=imread('flowers.tif');
[x,y,z]=sphere;          %定义球面
warp(x,y,z,RGB)
```

映射后的结果如图 4-10 所示。

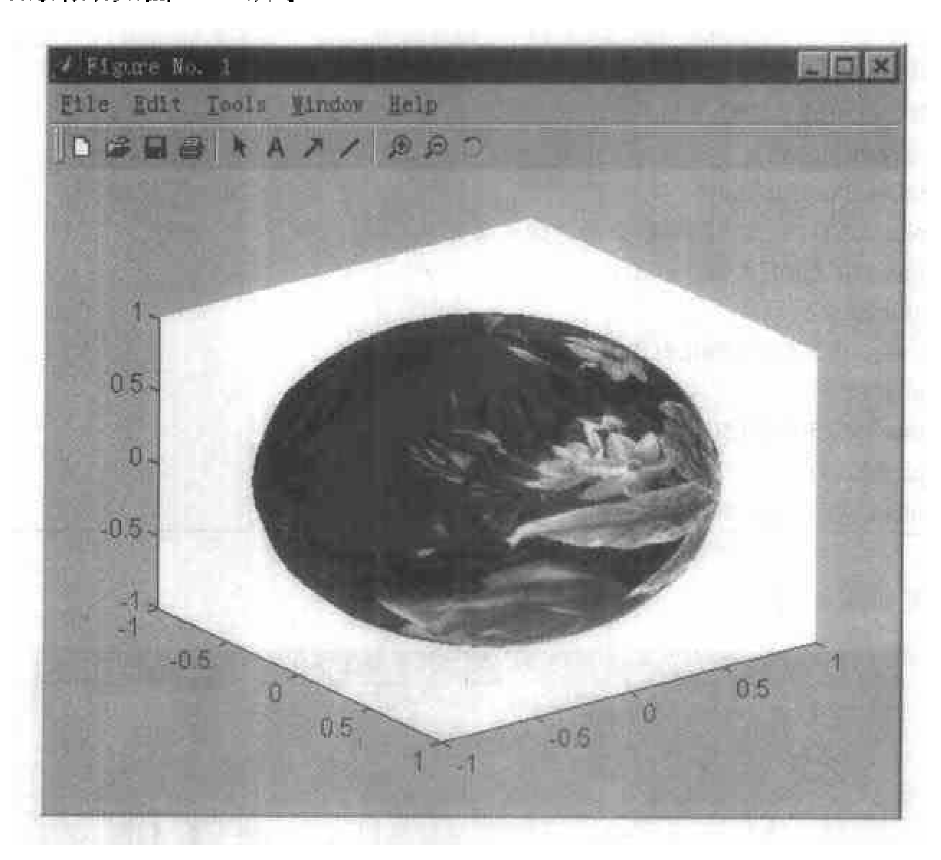


图 4-10　图像的纹理映射

4.2.5　在一个图形窗口中显示多幅图像

为了便于在多幅图像之间进行比较，我们需要将这些要比较的图像显示在一个图形窗口中。MATLAB 的图像处理工具箱就提供了这样一个函数，即 subimage 函数。下面具体介绍这个函数的格式和用法。

subimage 函数的调用格式如下：

```
subimage(X,map)
subimage(I)
subimage(RGB)
subimage(x,y,…)
h = subimage(…)
```

subimage(X,map)、subimage(I)和 subimage(RGB)分别用于显示索引色、灰度及真彩色图像。

subimage(x,y,…)表示将图像按指定的坐标系(x,y)显示。在具体应用时，主要是设置横

轴和纵轴的坐标值范围。

h = subimage(…)表示返回图像对象的句柄。

另外需要说明的是，subimage 函数必须与 subplot 函数一起使用，后者用于指定单个图像的位置。而且 subimage 函数所显示的图像必须是 uint8 或 double 类型。

例程 4-9 在一个图形窗口中同时显示 4 幅图像。

例程 4-9

```
RGB1=imread('pic005.jpg');
RGB2=imread('pic007.jpg');
I1=imread('rice.tif');
I2=imread('testpat1.tif');
subplot(2,2,1);    %将图形窗口等分成 4 部分
subimage([0,500],[0,500],RGB1);
subplot(2,2,2);
subimage([0,500],[0,500],RGB2);
subplot(2,2,3);
subimage([0,500],[0,500],I1);
subplot(2,2,4);
subimage([0,500],[0,500],I2);
```

显示结果如图 4-11 所示。

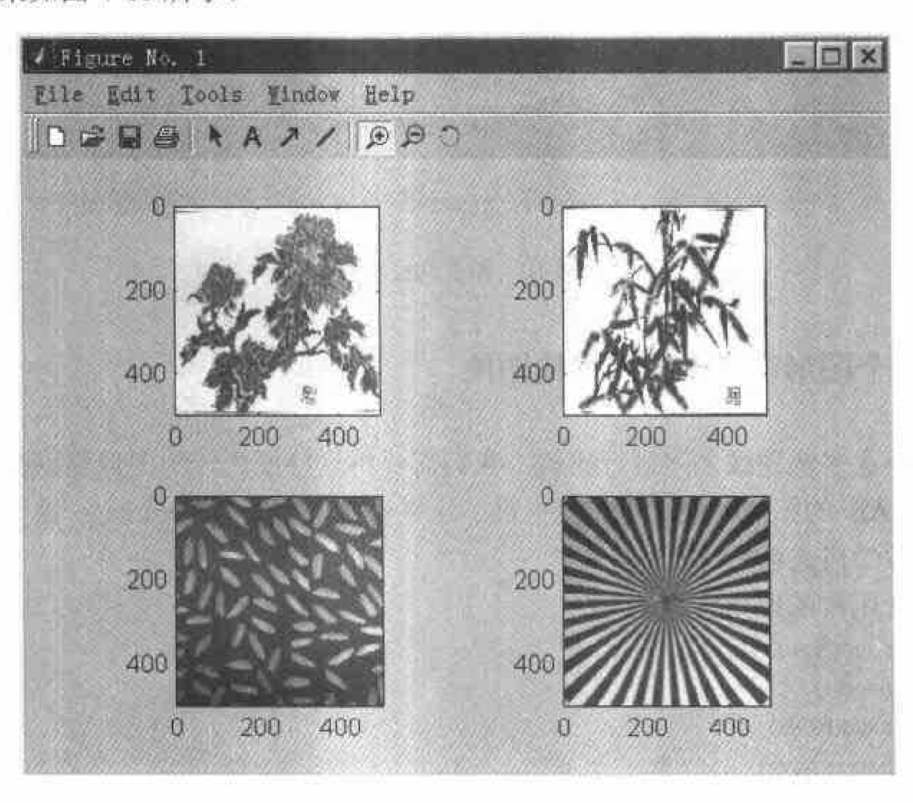


图 4-11　在一个窗口中显示多幅图像

4.3　MATLAB 中的颜色模型

在图像处理和显示的过程中，为了能够正确地使用颜色，需要建立颜色模型。颜色模型是三维颜色空间中的一个可见光子集，它包含某个颜色域的所有颜色。常用的颜色模型有 RGB、HSV、NTSC 和 YCbCr。

下面就具体介绍这几种模型及它们之间的相互转换关系。

4.3.1　颜色模型的分类

1. RGB 模型

RGB 模型是面向诸如彩色显示器或打印机之类的硬设备的常见的颜色模型。

该模型基于笛卡儿坐标系统，三个轴分别对应 R（即红色）、G（即绿色）、B（即蓝色）。为了方便，通常将该模型空间归一化为单位立方体，这样所用的 R、G、B 值都落在[0 1]区间。此时，原点对应黑色，离原点最远的立方体的顶点对应白色，而从黑到白的灰度值落在从原点到离原点最远的顶点的连线上。

2. HSV 模型

HSV 模型是面向用户的，是一种复合主观感觉的色彩模型。

HSV 模型对应于圆柱坐标系中的一个圆锥子集，如图 4-12 所示。HSV 模型中，H 即色调（Hue），表示颜色的种类，取值范围为[0 1]，相应的颜色从红、黄、绿、蓝绿、蓝、紫到黑变化，且它的值由绕 *V* 轴的旋转角决定，每一种颜色和它的补色之间相差 180°。S 即饱和度（Saturation），它的取值范围也是[0 1]，相应的颜色从未饱和（灰度）向完全饱和（无白色元素）变化。V 即亮度（Value），其取值范围同样是[0 1]，相应的颜色逐渐变亮。

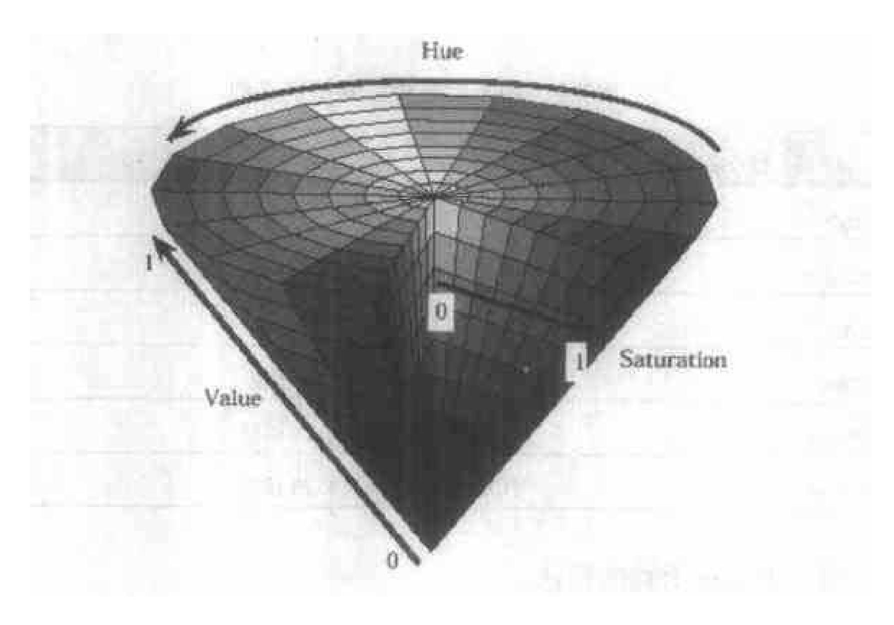


图 4-12　HSV 色彩模型

3．YCbCr 模型

YCbCr 模型是视频图像和数字图像中常用的色彩模型。

YCbCr 模型中，Y 为亮度，Cb 和 Cr 共同描述图像的色调，其中 Cb、Cr 分别为蓝色分量和红色分量相对于参考值的坐标。YCbCr 模型中的数据可以是双精度类型，但存储空间为 8 位无符号整型数据空间，且 Y 的取值范围为[16 235]，Cb 和 Cr 的取值范围为[16 240]。

4．NTSC 模型

NTSC 模型是一种用于电视图像的颜色模型。

NTSC 模型使用的是 Y.I.Q 色彩坐标系，其中 Y 为光亮度，I 为色调，Q 为饱和度。其中，亮度 Y 表示灰度信息，后两者表示颜色信息。因此，该模型的主要优点即将灰度信息和颜色信息区分开来。

Y.I.Q 坐标系与 R.G.B 坐标系的转换关系为：

$$\begin{pmatrix} R \\ G \\ B \end{pmatrix} = \begin{pmatrix} 1.000 & 0.956 & 0.621 \\ 1.000 & -0.272 & -0.647 \\ 1.000 & -1.106 & 1.703 \end{pmatrix} \begin{pmatrix} Y \\ I \\ Q \end{pmatrix}$$

$$\begin{pmatrix} Y \\ I \\ Q \end{pmatrix} = \begin{pmatrix} 0.299 & 0.587 & 0.114 \\ 0.596 & -0.274 & -0.322 \\ 0.211 & -0.523 & 0.312 \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix}$$

以上简单地介绍了各种颜色模型，下面介绍一下它们之间的相互转换关系。

4.3.2 颜色模型的转换

表 4-1 给出了 MATLAB 的图像处理工具箱中用于色彩模型转换的函数。

表 4-1 用于颜色模型转换的函数

函 数 名	函 数 功 能
hsv2rgb	HSV 模型转换为 RGB 模型
ntsc2rgb	NTSC 模型转换为 RGB 模型
rgb2hsv	RGB 模型转换为 HSV 模型
rgb2ntsc	RGB 模型转换为 NTSC 模型
rgb2ycbcr	RGB 模型转换为 YCbCr 模型
ycbcr2rgb	YCbCr 模型转换为 RGB 模型

下面就介绍一下这些函数的用法。

1．rgb2hsv 函数

该函数用来将 RGB 模型转换为 HSV 模型。其调用格式如下：

```
hsvmap = rgb2hsv(rgbmap)
HSV = rgb2hsv(RGB)
```

hsvmap = rgb2hsv(rgbmap) 将 RGB 空间中 $m\times3$ 的色彩表 rgbmap 转换成 HSV 色彩空间的颜色映射表 hsvmap。

HSV = rgb2hsv(RGB) 将真彩色图像 RGB 转换为 HSV 色彩空间中的图像 HSV。

例程 4-10 把一幅真彩图像转换为一个 HSV 模型空间对应的图像。

例程 4-10

```
RGB=imread('hr.jpg');
HSV=rgb2hsv(RGB);
subplot(1,2,1);
subimage(RGB);
title('原图像');
subplot(1,2,2);
subimage(HSV);
title('变换后的图像');
```

显示的结果如图 4-13 所示。



图 4-13 RGB 图像与 HSV 图像的比较

2. rgb2ntsc 函数

该函数用来将 RGB 模型转换为 NTSC 模型。其调用格式如下：

```
yiqmap = rgb2ntsc(rgbmap)
YIQ = rgb2ntsc(RGB)
```

yiqmap = rgb2ntsc(rgbmap)将 RGB 空间中 $m\times3$ 的色彩表 rgbmap 转换成 YIQ 空间中的颜色映射表 yiqmap。

YIQ = rgb2ntsc(RGB) 将真彩色图像 RGB 转换为 YIQ 色彩空间中的图像 YIQ。

例程 4-11 把一幅真彩图像转换为一个 NTSC 模型空间对应的图像（效果见彩插）。

例程 4-11

```
RGB=imread('hr.jpg');
YIQ=rgb2ntsc(RGB);
I=YIQ(:,:,1);
subplot(1,2,1);
subimage(RGB);
title('原图像');
subplot(1,2,2);
subimage(I);
title('变换后的图像');
```

执行的结果如图 4-14 所示。

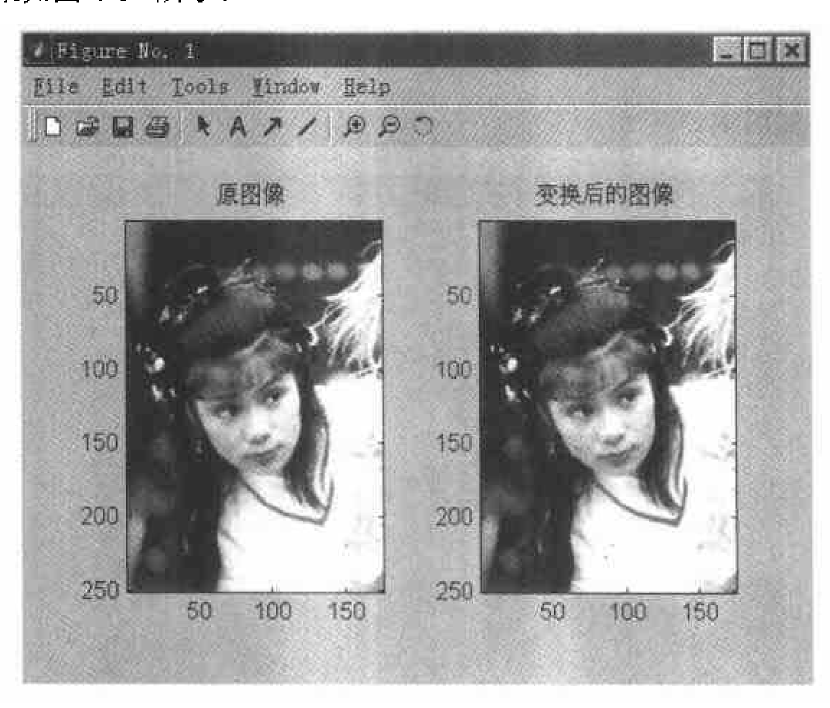


图 4-14 RGB 图像与 NTSC 图像的比较

3．rgb2ycbcr 函数

该函数用来将 RGB 模型转换为 YCbCr 模型。其调用格式如下：

ycbcrmap = rgb2ycbcr(rgbmap)

YCBCR = rgb2ycbcr(RGB)

ycbcrmap = rgb2ycbcr(rgbmap)将 RGB 空间中的色彩表 rgbmap 转换为 YCbCr 空间中的颜色映射表 ycbcrmap。

YCBCR = rgb2ycbcr(RGB)将真彩色图像 RGB 转换为 YCbCr 空间中的图像 YCBCR。

例程 4-12 把一幅真彩图像转换为一个 YCbCr 模型空间对应的图像。

例程 4-12

```
RGB=imread('hr.jpg');
YCbCr=rgb2ycbcr(RGB);
subplot(1,2,1);
subimage(RGB);
title('原图像');
subplot(1,2,2);
subplot(1,2,2);
subimage(YCbCr);
title('变换后的图像');
```

执行的结果如图 4-15 所示。



图 4-15　RGB 图像与 YCbCr 图像的比较

4．hsv2rgb 函数

该函数用来将 HSV 模型转换为 RGB 模型。其调用格式如下：

```
rgbmap = hsv2rgb(hsvmap)
RGB = hsv2rgb(HSV)
```

其中：rgbmap = hsv2rgb(hsvmap)表示将 HSV 色彩空间的颜色映射表 hsvmap 转换成 RGB 空间中的色彩表 rgbmap，rgbmap 和 hsvmap 都是 $m\times3$ 的矩阵。

RGB = hsv2rgb(HSV)表示将 HSV 色彩空间的图像 HSV 转换为真彩色图像 RGB。

5．ntsc2rgb 函数

该函数用来将 NTSC 模型转换为 RGB 模型。其调用格式如下：

```
rgbmap = ntsc2rgb(yiqmap)
RGB = ntsc2rgb(YIQ)
```

rgbmap = ntsc2rgb(yiqmap)表示将 YIQ 空间中 $m\times3$ 的颜色映射表 yiqmap 转换成 RGB 空间中的色彩表 rgbmap。

RGB = ntsc2rgb(YIQ) 表示将 YIQ 色彩空间的图像 YIQ 转换为真彩色图像 RGB。

6．ycbcr2rgb 函数

该函数用来将 YCbCr 模型转换为 RGB 模型。其调用格式如下：

```
rgbmap = ycbcr2rgb(ycbcrmap)
RGB = ycbcr2rgb(YCBCR)
```

rgbmap = ycbcr2rgb(ycbcrmap)将 YCbCr 空间中的颜色映射表 ycbcrmap 转换为 RGB 空间中的色彩表 rgbmap。

RGB = ycbcr2rgb(YCBCR) 将 YCbCr 空间中的图像 YCBCR 转换为真彩色图像 RGB。

第5章　MATLAB图像几何操作和处理

基于区域的图像处理本章对图像的几种几何操作和对图像的特定区域的处理进行较为详细的介绍，使读者能够了解相关的原理，并且运用MATLAB来实现上述的操作。

本章主要内容：

- 图像的缩放
- 图像的旋转
- 图像的剪切

5.1　图像插值的基本原理

在对图像的几种几何操作中，图像的缩放和旋转都要用到插值操作。插值算法的好坏直接关系到图像的失真程度，插值函数的设计是插值算法的核心问题。因此，在具体介绍这几种几何操作前，有必要先介绍一下插值的基本原理。

插值通常是利用曲线拟合的方法，通过离散的采样点建立一个连续函数，用这个重建的函数便可以求出任意位置的函数值，如图5-1所示。

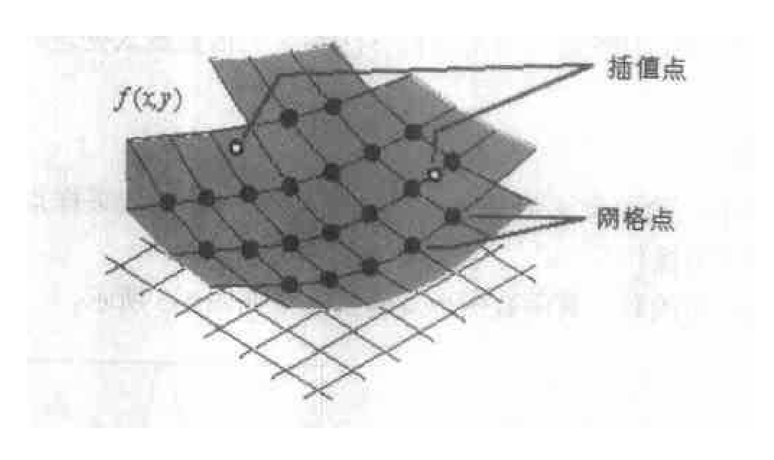


图5-1　图像的插值

对于等间隔离散数据，插值可以表示为

$$f(x)=\sum_{k=0}^{K-1}C_k h(x-x_k)$$

其中，h为插值核，C_k为权系数。

插值算法的数值精度及计算开销直接与插值核有关，插值核的设计是插值算法的核心。MATLAB的图像处理工具箱提供了三种插值方法：

- 最近邻插值（Nearest neighbor interpolation）
- 双线性插值（Bilinear interpolation）
- 双三次插值（Bicubic interpolation）

下面具体介绍这几种插值方法的原理和特点。

1．最近邻插值

从计算量的角度来说，最近邻插值是最简单的插值。在这种算法中，每一个插值输出像素的值就是在输入图像中与其最临近的采样点的值。算法的数学表示为：

$$f(x)=f(x_k) \quad , \quad \frac{1}{2}(x_{k-1}+x_k)<x\leqslant\frac{1}{2}(x_k+x_{k+1})$$

最近邻法的插值核及其傅里叶谱如图 5-2 所示。

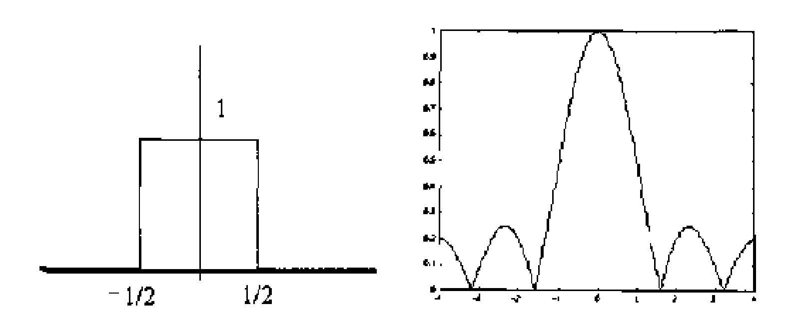


图 5-2　最近邻法的插值核及其傅里叶谱

最近邻插值是工具箱函数默认使用的插值方法。对于索引色图像来讲，它也是惟一可行的方法。不过，最近邻法插值核的频域特性并不好，从它的傅里叶谱上可以看出，它与理想低通滤波器的性质相差较大。当用这种方法实现大倍数放大处理时，在图像中可以明显地看出块状效应。

2．双线性插值

在双线性插值中，输出像素值是它在输入图像中 2×2 邻域采样点的平均值。这样看来，它相当于一种平滑操作。

它的插值核为三角函数，其函数图形及傅里叶谱如图 5-3 所示。

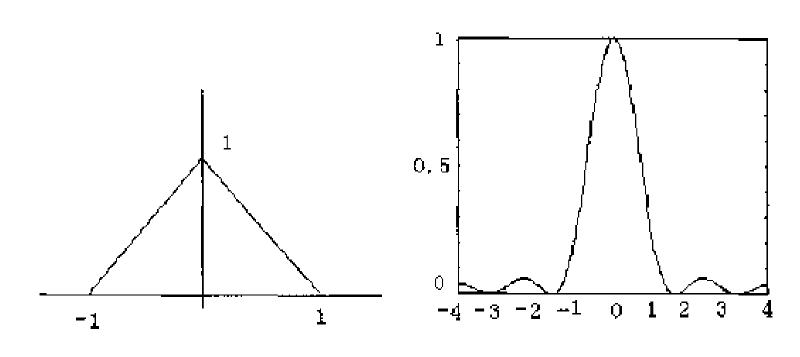


图 5-3　线性插值的插值核及其傅里叶谱

从其傅里叶谱上可以看出它的频域特性优于最近邻插值函数，其频谱的旁瓣远小于主



瓣，表明它的带阻特性较好。不过，仍有大量高频成分漏入通频带，造成了一定的混叠。此外，通频带在一定程度上被减弱，会使插值后的图像变模糊，从而损失一些细节。

3. **双三次插值**

双三次插值的插值核为三次函数，其插值邻域的大小为 4×4。它的插值效果比较好，但相应的计算量也较大，一般很少使用。

双三次插值法的插值核及其傅里叶谱如图 5-4 所示

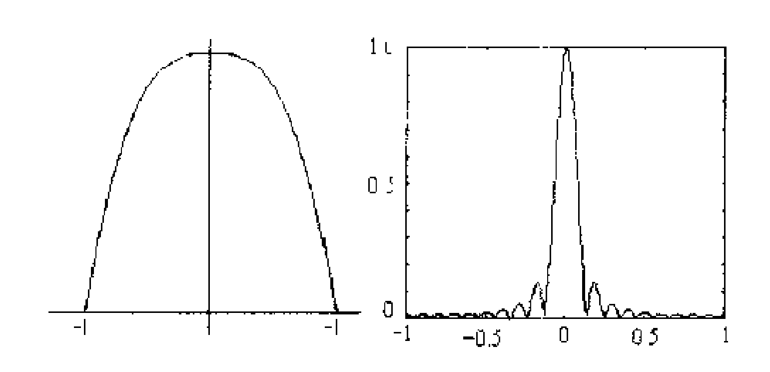


图 5-4 三次插值的插值核及其傅里叶谱

5.2 图像的插值缩放

在 MATLAB 中，用函数 imresize 来实现对图像的放大或缩小。插值方法可选用以上三种方法中的任意一种，默认的插值方法是最近邻法。

该函数的调用格式如下：

```
B = imresize(A,m,method)
B = imresize(A,[mrows ncols],method)
B = imresize(…,method,n)
B = imresize(…,method,h)
```

参数 method 用于指定插值的方法，可选的值为'nearest'（最近邻法）、'bilinear'（双线性插值）、'bicubic'（双三次插值），默认值为'nearest'。

B = imresize(A,m,method)返回原图 A 的 m 倍放大图像（m 小于 1 时实际上是缩小）。

B = imresize(A,[mrows ncols],method)返回一个 mrows 行、ncols 列的图像，若与原图的长/宽比不同，则图像会产生变形。

在使用'bilin'和'bicubic'方法缩小图像时，为消除引入的高频成分，imresize 使用了一个前端平滑滤波器，默认的滤波器尺寸为11×11。用户也可通过参数 n 指定滤波器的尺寸，即 B = imresize(…,method,n)。

B = imresize(…,method,h)使用用户设计的插值核 h 进行插值，h 可以看做一个二维FIR滤波器。

例程 5-1 使用不同插值方法放大图像。

例程 5-1

```
I=imread('rice.tif');
imshow(I);
I1=imresize(I,1.5,'nearest neighber');    %最近邻插值
figure,imshow(I1);
I2=imresize(I,1.5,'bilinear'); %双线性插值
figure,imshow(I2);
I3=imresize(I,1.5,'bicubic'); %双三次插值
figure,imshow(I3);
```

图 5-5 为原图像，图 5-6、图 5-7 和图 5-8 采用了上述三种不同的方法放大图像。

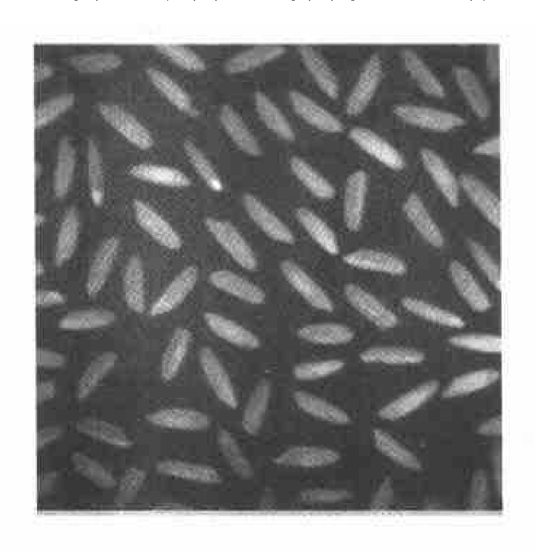


图 5-5　原图像

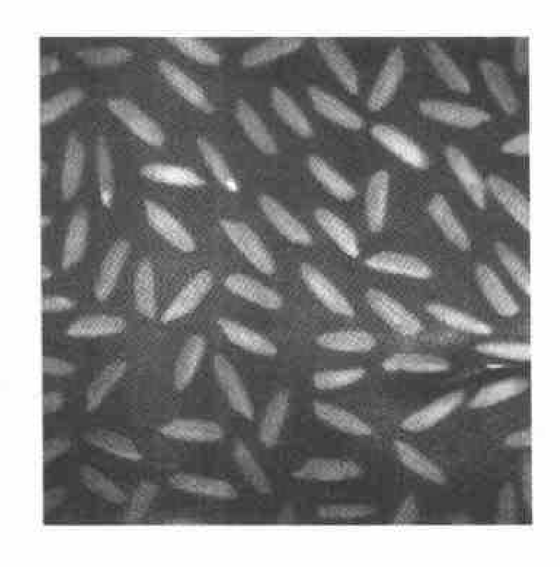


图 5-6　近邻插值法放大 1.5 倍图像的结果

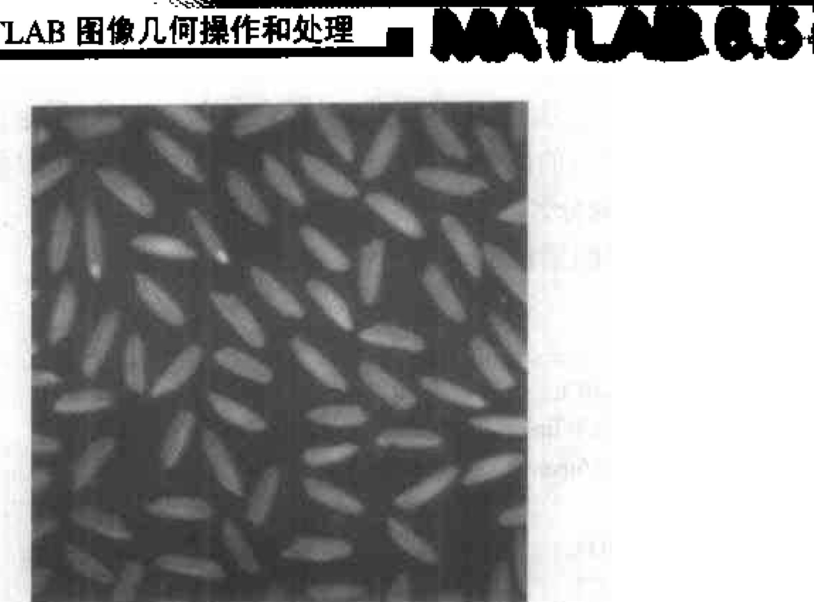


图 5-7 双线性插值法放大 1.5 倍图像的结果

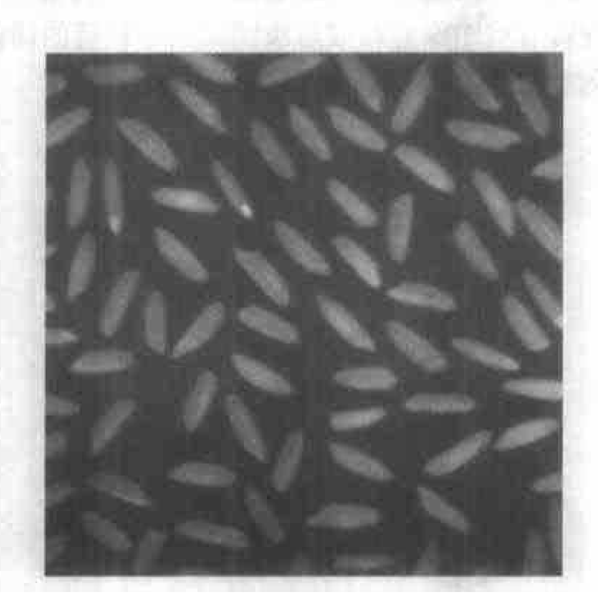


图 5-8 双三次插值法放大 1.5 倍图像的结果

由以上各图可看出，最近邻插值的效果是最差的，双三次插值是最好的，双线性插值的效果介于两者之间。

5.3 图像的插值旋转

在 MATLAB 中，用函数 imrotate 来实现对图像的插值旋转，插值方法也可选用三种方法中的任意一种，默认的插值方法是最近邻法。

该函数的调用格式如下：

```
B = imrotate(A,angle,method)
B = imrotate(A,angle,method,'crop')
```

参数 method 用于指定插值的方法，可选的值为'nearest'（最近邻法）、'bilinear'（双线

性插值)、'bicubic'（双二次插值)，默认值为'nearest'。参数 angle 代表旋转的角度。

一般来说，旋转后的图像会比原图大，用户可以指定'crop'参数对旋转后的图像进行剪切（取图像的中间部分)，使返回的图像与原图大小相同。

例程5-2表示图像的插值旋转。

例程 5-2

```
I=imread('eight.tif');
I1=imrotate(I,30,'bilinear','crop');
I2=imrotate(I,30,'nearest neighber');
imshow(I);
figure,imshow(I1);
figure,imshow(I2);
```

将图像eitht.tif旋转了30°，I2是采用最近邻插值，且旋转后没有剪裁图像，而I1则是采用的双线性插值，且旋转后将图像剪裁成与原图像大小相等的图像。

原图及I1和I2的显示结果分别如图5-9、图5-10及图5-11所示。

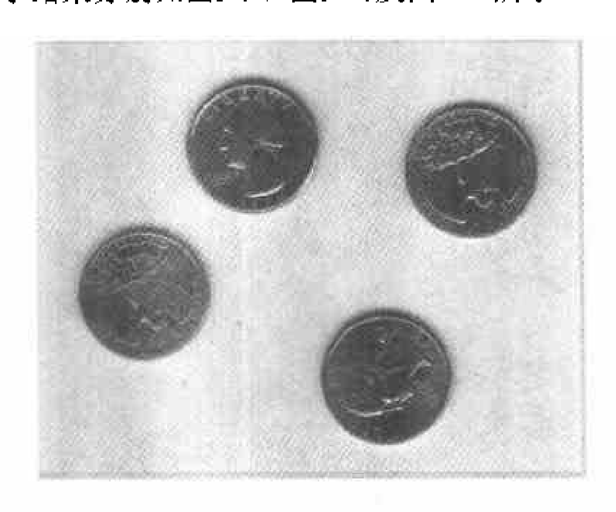


图 5-9　原图像

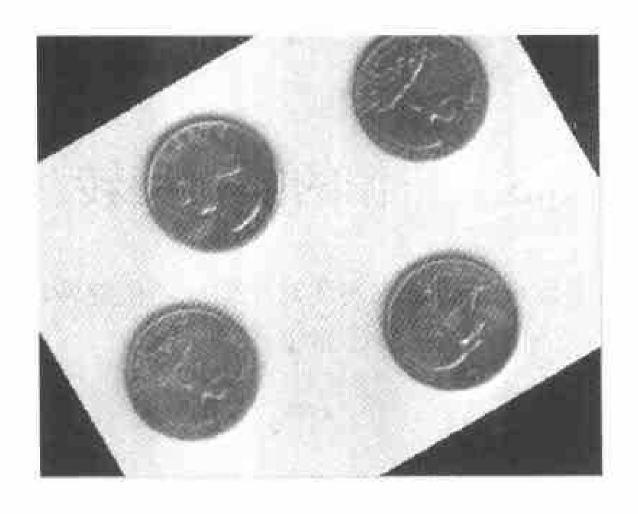


图 5-10　旋转 30° 后的图像（带'crop'参数）

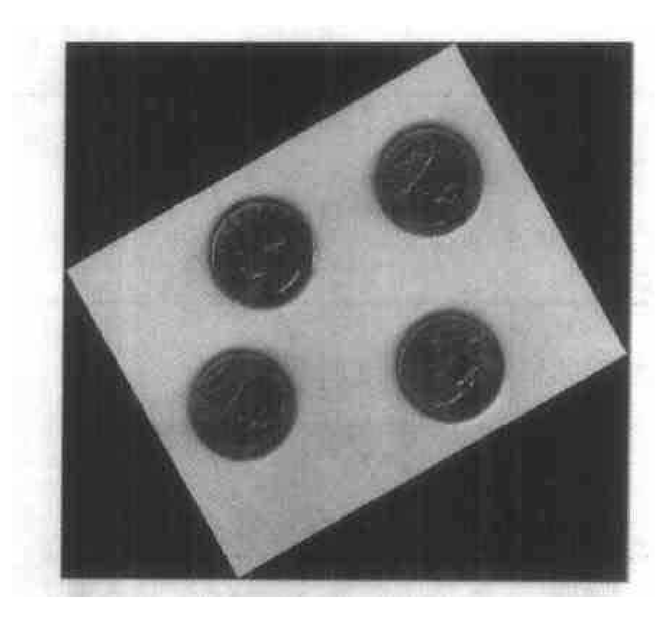


图 5-11　旋转 30° 后的图像（不带'crop'参数）

5.4　图像的剪切

MATLAB 中，用函数 imcrop 实现对图像的剪切操作。该操作剪切的是图像中的一个矩形子图，用户可以通过参数指定这个矩形四个顶点的坐标，也可以交互地用鼠标选取这个矩形。

imcrop函数的调用格式如下：

```
I2 = imcrop(I)
X2 = imcrop(X,map)
RGB2 = imcrop(RGB)
I2 = imcrop(I,rect)
X2 = imcrop(X,map,rect)
RGB2 = imcrop(RGB,rect)
[…] = imcrop(x,y,…)
[A,rect] = imcrop(…)
x,y,A,rect] = imcrop(…)
```

I2 = imcrop(I)、X2 = imcrop(X,map)和 RGB2 = imcrop(RGB) 分别表示交互式地对灰度图像、索引色图像和真彩色图像进行剪切操作。

I2 = imcrop(I,rect) 、X2 = imcrop(X,map,rect)和 RGB2 = imcrop(RGB,rect)分别表示按指定的矩形框 rect 剪切图像，rect 是一个 4 元矢量[xmin ymin widt height]。

[…] = imcrop(x,y,…)表示在指定坐标系(x,y)中剪切图像。

[A,rect] = imcrop(…)和[x,y,A,rect] = imcrop(…)表示在剪切图像的同时返回剪切框的参数 rect。

例程 5-3 从图像 flower.tif 中剪切了一块子图。

例程 5-3

```
%顶点坐标为(40 , 50) ， (40,250)，（240,50）和（240,250）。
RGB=imread('flowers.tif');
imshow(RGB);
RGB2=imcrop(RGB,[40,50,200,200]);
figure,imshow(RGB2);
```

原图见图 5-12，剪切后的结果如图 5-13 所示。



图 5-12　原图像



图 5-13　剪切后的图像

5.5　基于区域的图像处理

要对一幅图像的特定的区域进行处理时，首先要定义这个我们感兴趣的区域。在 MATLAB 中，对这个特定区域的定义是通过创建一个二进制的 mask（为一幅二进制图像）来实现的，暂且称其为 mask 图像。该 mask 图像与原图像具有相同的尺寸，我们所选定的区域的像素在 mask 中的值为 1（即为白色），其余部分的像素值为 0（即为黑色）。这样就通过 mask 图像，实现了对特定区域的选择。

下面就具体介绍如何创建一个 mask 图像，来实现特定区域的选择。



5.5.1　多边形选择法

在 MATLAB 中，用函数 roipoly 可以设定一个多边形区域。其调用格式如下：

```
BW = roipoly(I,c,r)
BW = roipoly(I)
BW = roipoly(x,y,I,xi,yi)
[BW,xi,yi] = roipoly(···)
[x,y,BW,xi,yi] = roipoly(···)
```

roipoly 函数返回二值图像 BW，选中的区域为白色，其余的部分为黑色。

BW = roipoly(I,c,r)是用矢量 c、r 来指定多边形各顶点的 *X*、*Y* 轴的坐标。BW = roipoly(I)是让用户交互地选择多边形区域，用鼠标左键选择顶点，用空格键或【Delete】键撤消选择，按回车键确认。

BW = roipoly(x,y,I,xi,yi)是在指定的坐标系 *XY* 下选择由矢量 xi、yi 指定的多边形区域。

[BW,xi,yi] = roipoly(···)交互地选择多边形区域，并返回多边形各顶点的坐标。

[x,y,BW,xi,yi] = roipoly(···)在交互地选择多边形区域后，返回多边形的各顶点在指定的坐标系 *XY* 下的坐标。

例程 5-3 选择图像中的六边形区域。

例程 5-3

```
I = imread('eight.tif');
c = [222 272 300 270 221 194];  %定义多边形顶点的 x 坐标
r = [21 21 75 121 121 75];  %定义多边形顶点的 y 坐标
BW = roipoly(I,c,r);
imshow(I);
figure;
imshow(BW);
```

其原图像与 mask 图像如图 5-14 和图 5-15 所示。

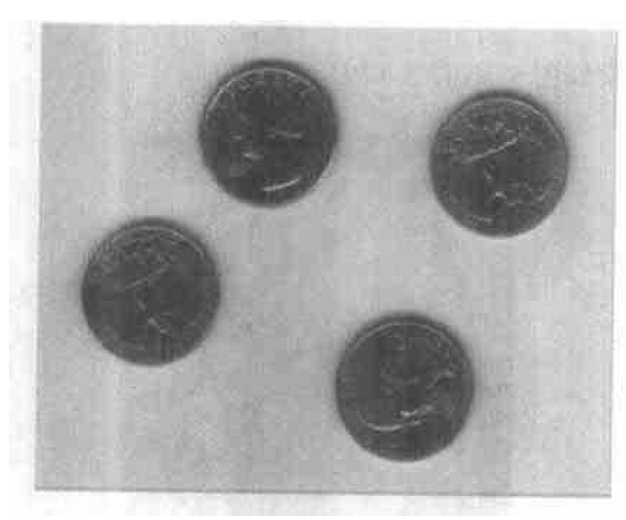


图 5-14　原图像

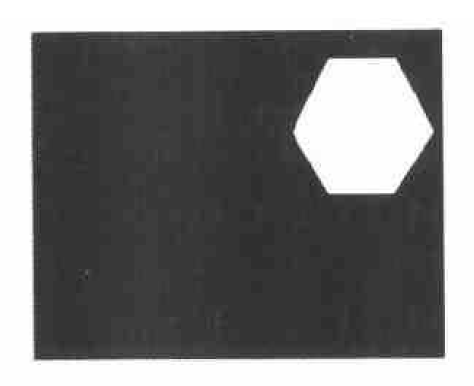


图 5-15　mask 图像

5.5.2　灰度选择法

在 MATLAB 中，用户可以用 roicolor（）函数来根据颜色和灰度范围设定感兴趣的区域。

roicolor（）函数的调用格式如下：

```
BW = roicolor(A,low,high)
BW = roicolor(A,v)
```

BW = roicolor(A,low,high)按指定的灰度范围分割图像，返回代表 mask 图像的数据矩阵 BW，[low high]为所要选择区域的灰度范围。

BW = roicolor(A,v)是按矢量 v 中指定的灰度值来选择区域。

例程 5-4 选中灰度值在[150 200]之间的图像中的区域。

例程 5-4

```
I = imread('rice.tif');
BW = roicolor(I,150,200);
imshow(I);
figure;
imshow(BW)
```

原图及选择后的结果如图 5-16 和 5-17 所示。

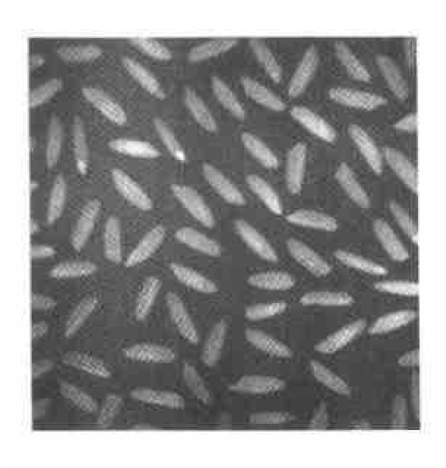


图 5-16　原图像

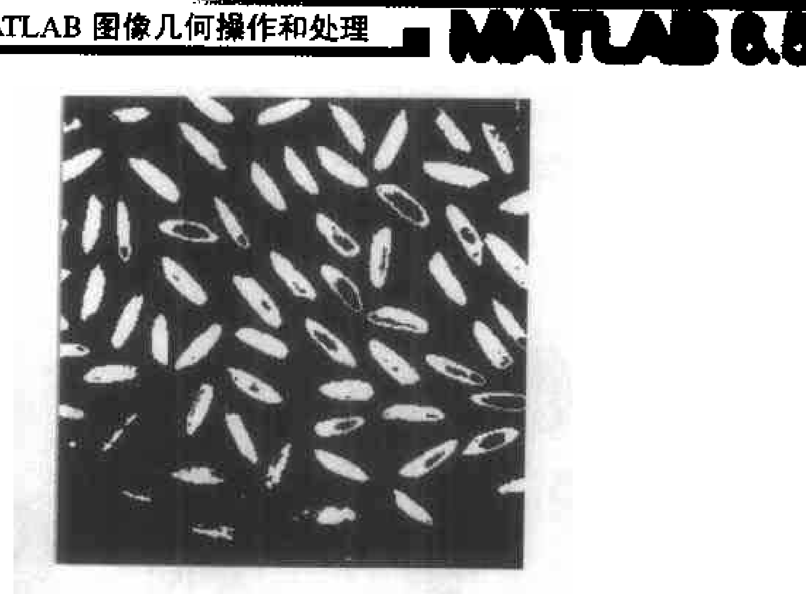


图 5-17 按灰度选择后的区域

5.5.3 其他选择方法

除了使用上述两种方法生成mask图像外，用户还可以选用任意的二值图像来作为mask图像，但该二值图像必须具有与原始图像相同的尺寸。

想对一个数据矩阵为 I 的灰度图像进行过滤，过滤时只滤去灰度值大于 200 的像素。则我们可以用下面的命令生成一个合适的 mask：

```
I=imread('rice.tif');
BW=(I>200);
```

5.5.4 对指定区域的滤波

在 MATLAB 中，用函数 roifilt2 实现对指定区域的滤波。其调用格式如下：

```
J = roifilt2(h,I,BW)
J = roifilt2(I,BW,fun)
J = roifilt2(I,BW,fun,P1,P2, …)
```

J = roifilt2(h,I,BW)表示使用滤波器 h 对图像 I 中用 mask 选中的区域进行滤波。

J = roifilt2(I,BW,fun) 和 J = roifilt2(I,BW,fun,P1,P2,…) 表示对图像 I 中用 mask 选中的区域作函数运算 fun，其中 fun 是描述函数运算的字符串，参数为 P1,P2,…。返回图像 J 在选中区域的像素为 I 经 fun 运算的结果，其余部分的像素值为 I 的原始值。

例程 5-5 对指定区域进行锐化滤波。

例程 5-5

```
I = imread('eight.tif');
c = [222 272 300 270 221 194];
r = [21 21 75 121 121 75];
BW = roipoly(I,c,r);
h = fspecial('unsharp');
J = roifilt2(h,I,BW);
```

```
subplot(1,2,1),imshow(I);
subplot(1,2,2),imshow(J);
```

结果如图 5-18 所示。

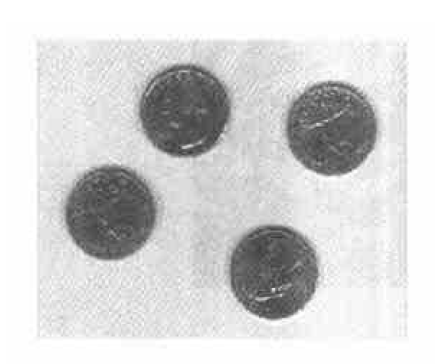
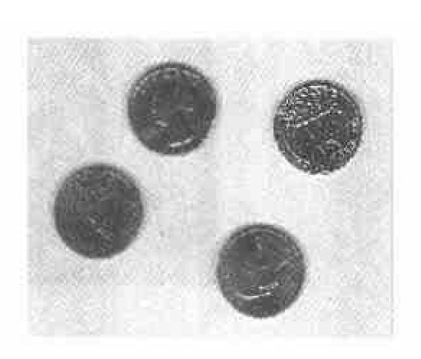



图 5-18 对指定区域的滤波

5.5.5 对指定区域的填充

在 MATLAB 中，用函数 roifill 实现对指定区域的填充。其调用格式如下：

```
J = roifill(I,c,r)
J = roifill(I)
J = roifill(I,BW)
[J,BW] = roifill(…)
J = roifill(x,y,I,xi,yi)
[x,y,J,BW,xi,yi] = roifill(…)
```

J = roifill(I,c,r)表示填充由矢量 c、r 指定的多边形，c 和 r 分别为多边形各顶点的 *X*、*Y* 轴的坐标。

J = roifill(I)表示由用户交互式地选取填充的区域。用鼠标左键选择多边形的各顶点，按回车键表示结束，按空格键或【Delete】键表示取消一个选择。

J = roifill(I,BW)表示填充由 BW 决定的 mask 图像所选择的区域。

[J,BW] = roifill(…)在填充区域的同时还返回代表 mask 图像的数据矩阵 BW。

J = roifill(x,y,I,xi,yi)和[x,y,J,BW,xi,yi] = roifill(…)在指定的坐标系 *XY* 下填充由矢量 xi、yi 指定的多边形区域。

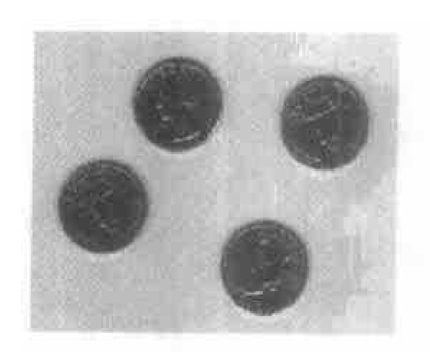
例程 5-6 填充一个指定的区域。

例程 5-6

```
I = imread('eight.tif');
c = [222 272 300 270 221 194];
r = [21 21 75 121 121 75];
J = roifill(I,c,r);
subplot(1,2,1),imshow(I)
subplot(1,2,2),imshow(J)
```



填充的结果如图 5-19 所示。



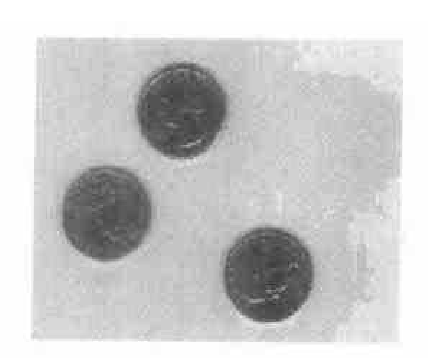


图 5-19 对指定区域的填充

第6章 MATLAB图像变换

为了快速有效地对图像进行处理和分析，比如进行图像增强、图像分析、图像复原、图像压缩等，常常需要将原定义在图像空间的图像以某种形式转换到另外一些空间，并利用在这些空间的特有性质方便地进行一定的加工，最后再转换回图像空间以得到所需的效果。因此，图像变换是许多图像处理和分析的基础。

本章主要内容：

- 二维连续傅里叶变换
- 二维离散傅里叶变换
- 快速傅里叶变换
- 离散余弦变换
- Radon 变换

6.1 傅里叶变换及实现

傅里叶变换是线性系统分析的一个有力工具，它将图像从空域变换到频域，使我们能够定量地分析诸如数字化系统、采样点、电子放大器、卷积滤波器、噪声、显示点等的作用（效应），把傅里叶变换的理论同其物理解释相结合，将大大有助于解决大多数图像处理问题。

在数字图像处理中，输入图像和输出图像通常都是二维的，一般表示成二维数字矩阵，因此这里直接讨论二维傅里叶变换、二维 DFT、二维 FFT

6.1.1 二维连续傅里叶变换

二维傅里叶变换的定义如下：

$$F(u,v)=\sum_{x=-\infty}^{\infty}\sum_{y=-\infty}^{\infty}f(m,n)\mathrm{e}^{-jum}\mathrm{e}^{-jvn}$$

其中 u 和 v 是频率变量，其单位是弧度/每采样单位；$F(u,v)$ 通常称为 $f(x,y)$ 的频域表示。$F(u,v)$ 是复值函数，在 u 和 v 上都是周期性的，且周期为 2π。因为其周期性，通常只显示 $-\pi\leqslant u,v\leqslant\pi$ 的范围。注意，$F(0,0)$ 是 $f(x,y)$ 的所有值之和，因此，$F(0,0)$ 通常称为傅里叶变换的恒定分量或 DC 分量（DC 表示直流，是电子工程学

术语，指电压恒定的电源，与电压值呈正弦变化的电源不同）。如果 $f(x,y)$ 是一幅图像，则 F（u，v）是它的谱，变量 u 对应着 X 轴，频率 v 对应着 Y 轴。

二维傅里叶反变换定义如下：

$$f(x,y)=\frac{1}{2\pi}\int_{u=-\pi}^{\pi}\int_{v=-\pi}^{\pi}F(u,v)\mathrm{e}^{jux}\,\mathrm{e}^{jvy}\,\mathrm{d}u\,\mathrm{d}v$$

粗略地讲，此方程表明 f（x，y）可以表示为无穷多个不同频率的复指数幂（正弦的）之和，在频率（u，v）上贡献的幅值和相位由 F（u，v）给出。

同时定义二维傅里叶变换的频谱、相位谱和功率谱（频谱密度）如下：

$$|F(u,v)|=\left[R^2(u,v)+I^2(u,v)\right]^{\frac{1}{2}}$$

$$\Phi(u,v)=\arctan\left[I(u,v)/R(u,v)\right]$$

$$P(u,v)=|F(u,v)|^2=R^2(u,v)+I^2(u,v)$$

其中 R（u,v）和 I（u,v）分别为 F（u,v）的实部和虚部。

由于幅度谱中表现了一些可辨认的结构，而相位谱看起来是完全随机的，所以一般可能认为幅度谱比较重要，但事实却恰恰相反。如果忽略相位信息，（将相位设为零）然后实行反变换，从得到的图 6-4 中完全看不出来与原图有什么相似之处。如果忽略幅值信息（在进行反变换之前将幅值设为常数），从得到的图 6-5 中，还可以辨认出肖像的轮廓。这说明相位谱在数字图像处理中有着更重要的潜在作用，在处理图像的过程中一定不能忽略了相位谱的作用。



图 6-1　原始图像

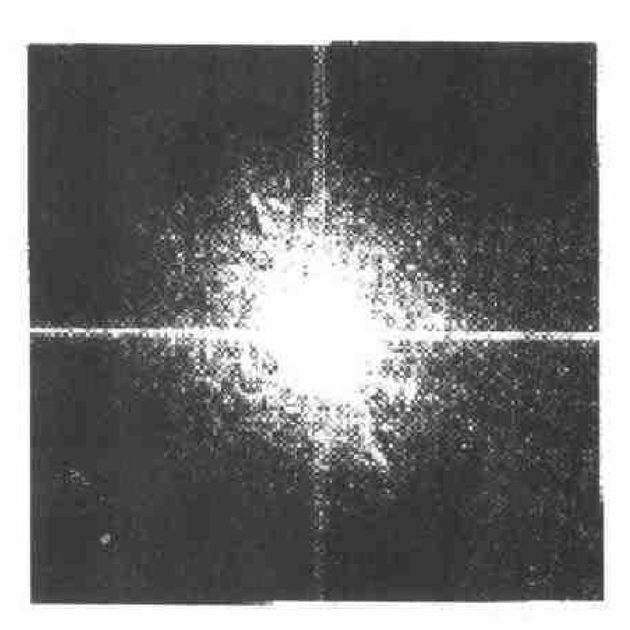


图 6-2　幅值谱

幅值谱表明了各正弦分量出现的多少，而相位信息表明了各正弦分量在图像中出现的位置。图 6-1、图 6-2、图 6-3、图 6-4、图 6-5 表明打乱了各正弦分量出现的位置将造成毁灭性的影响。但是只要各正弦分量保持原位，则对于图像整体来说，幅值就显得不那么重要。出于这些原因，大多数实用滤波器都只影响幅值，而几乎不改变相位信息。

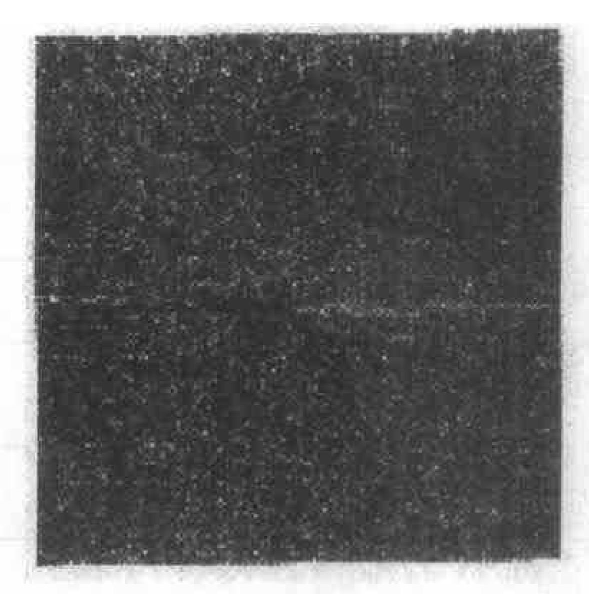


图 6-3 相位谱

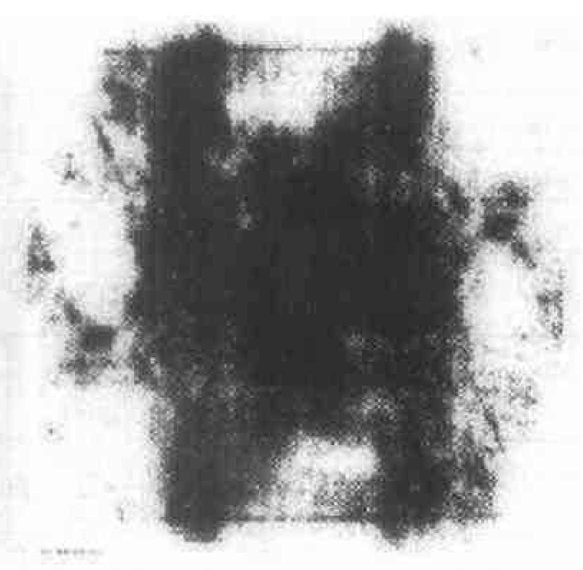


图 6-4 由幅值谱重构的图像



图 6-5 由相位谱重构的图像

二维傅里叶变换有很多重要性质，例如可分离性、平移性、周期性、共轭对称性、旋转性、满足分配律、可进行尺度变换（缩放）、卷积等等，这些性质使得在频域中对图像进行操作变得很简单，从而简化了整个处理过程，因此在图像处理中都有重要应用。表 6-1 列出了其中的一些性质。

表 6-1 傅里叶变换的性质

性质	空域	频域
加法定理	$f(x,y)+g(x,y)$	$F(u,v)+G(u,v)$
相似性定理	$f(ax,by)$	$\frac{1}{\lvert ab\rvert}F(\frac{u}{a},\frac{v}{b})$
位移定理	$f(x-a,y-b)$	$e^{-j2\pi(au+bv)}F(u,v)$
卷积定理	$f(x,y)\times g(x,y)$	$F(u,v)G(u,v)$
可分离乘积	$f(x)g(y)$	$F(u)G(v)$
微分	$(\frac{\partial}{\partial x})^m(\frac{\partial}{\partial y})^n f(x,y)$	$(j2\pi u)^m(j2\pi v)^n F(u,v)$
旋转	$f(x\cos\theta+y\sin\theta,$ $-x\sin\theta+y\cos\theta)$	$F(u\cos\theta+v\sin\theta,$ $-u\sin\theta+v\cos\theta)$
拉普拉斯	$\nabla^2 f(x,y)=\frac{\partial^2}{\partial x^2}+\frac{\partial^2}{\partial y^2}f(x,y)$	$-4\pi^2(u^2+v^2)F(u,v)$
Rayleigh 定理	$\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\lvert f(x,y)\rvert^2\,\mathrm{d}x\,\mathrm{d}y=\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\lvert F(u,v)\rvert^2\,\mathrm{d}u\,\mathrm{d}v$	

1. **平移性**

由位移定理可以看出，$f(x,y)$ 的平移不影响其傅里叶变换的幅值，不管图像沿 X 轴平移，沿 Y 轴平移，还是在两个方向上都有平移，对应的幅值谱都和原始图像是完全一样的，如图 6-6、图 6-7、图 6-8、图 6-9、图 6-10、图 6-11 所示。根据此性质，当对图像进行傅里叶变换后，如果原点不在图像的中心，则可以先进行平移，对平移后的幅值谱进行处理，然后再将处理后的幅值谱平移回原位。这样操作不但对得到的结果没有影响，还省掉了幅值谱的原点不在图的中心带来的麻烦。

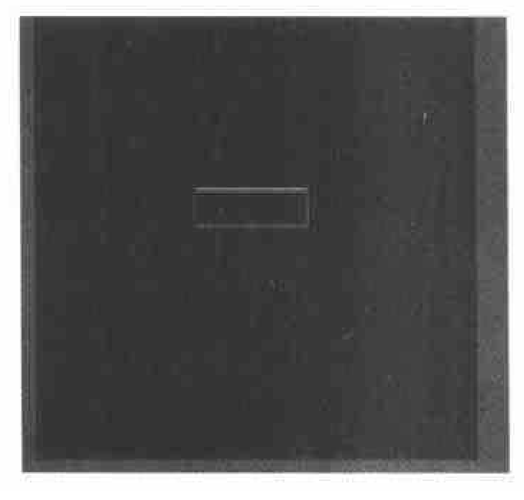


图 6-6 原始图像

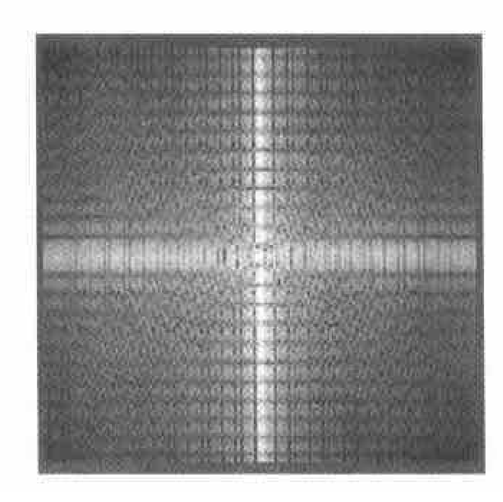


图 6-7 原图的傅里叶谱



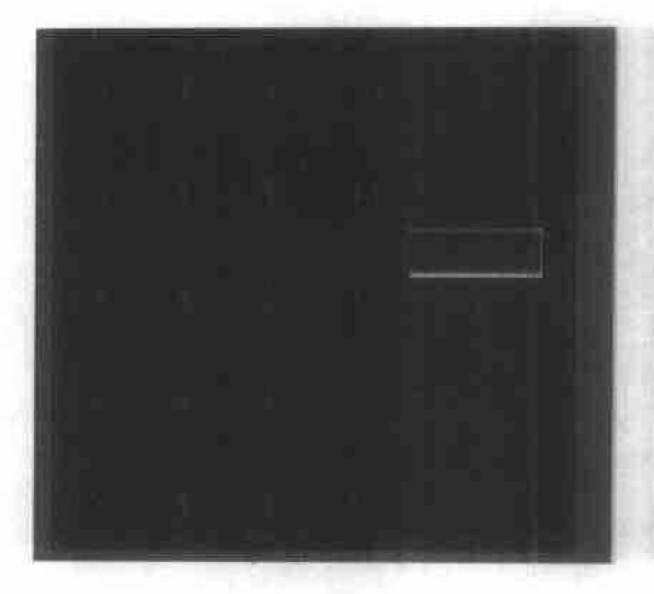


图 6-8　*X* 轴平移图像

图 6-9　*X* 轴平移图像的傅里叶谱

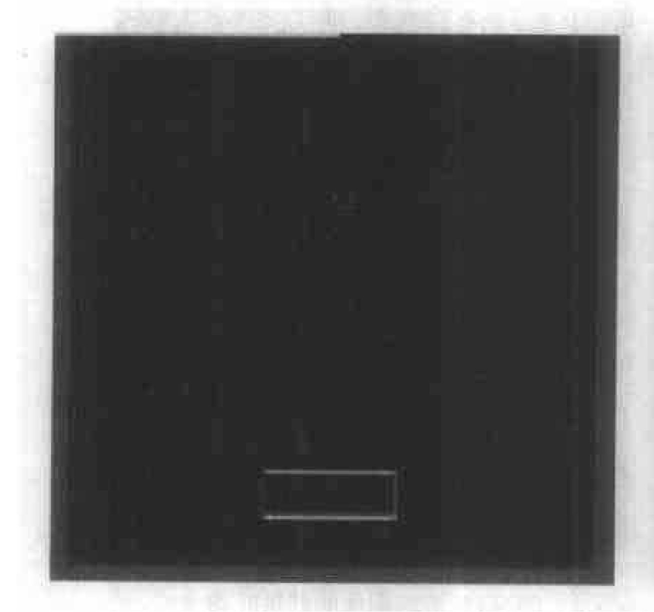


图 6-10　*Y* 轴平移图像

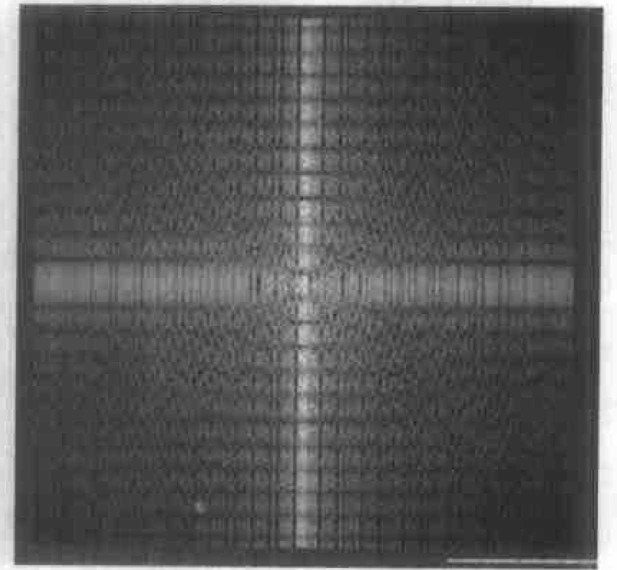


图 6-11　*Y* 轴平移图像的傅里叶谱

2．旋转性质

由列表可知，$f(x,y)$ 旋转 θ 角，则其傅里叶变换 $F(u,v)$ 也旋转 θ 角；$F(u,v)$ 旋转 θ 角，则 $f(x,y)$ 也旋转 θ 角。如图 6-12、图 6-13、图 6-14、图 6-15 所示，将原始图像旋转了 45°，而相应的傅里叶谱也旋转了 45°。

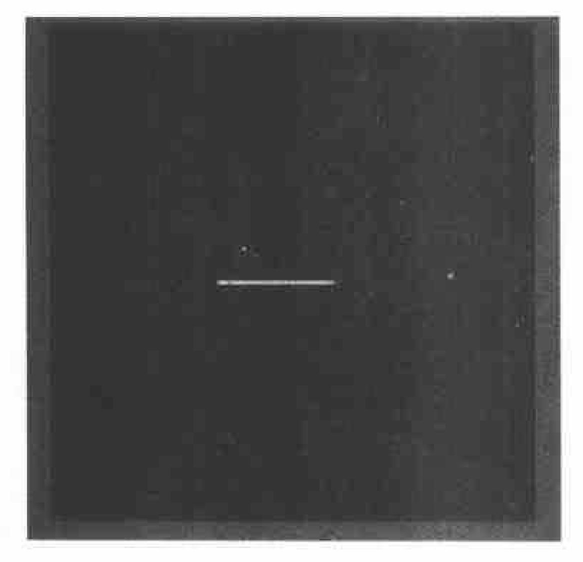


图 6-12　原始图像

图 6-13　原图的傅里叶谱

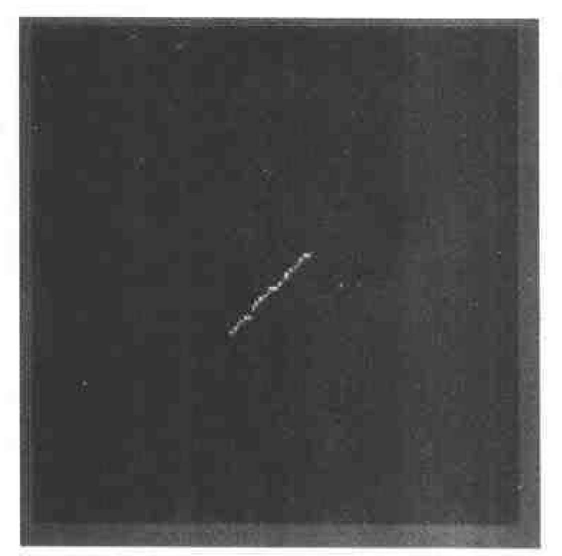


图 6-14 旋转后的图像

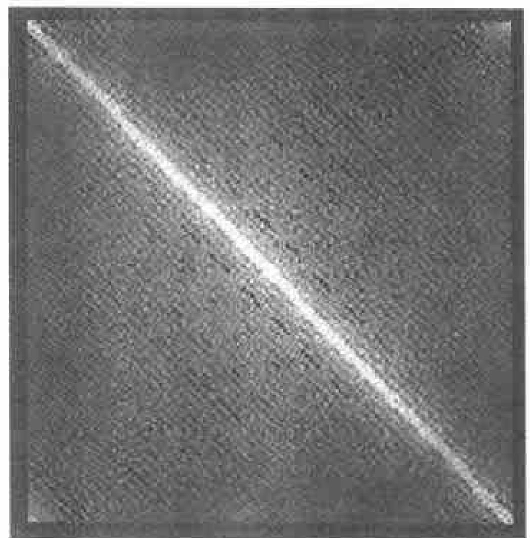


图 6-15 旋转图像的傅里叶谱

6.1.2　二维离散傅里叶变换（DFT）

二维 M×N 的 DFT 变换和逆 DFT 变换分别定义如下：

$$F(m,n)=\sum_{i=0}^{M-1}\sum_{k=0}^{N-1} f(i,k)\,e^{-j(2\pi/M)mi}\,e^{-j(2\pi/N)nk}\qquad p=0,1,\cdots,M\text{-}1$$

$$q=0,1,\cdots,N\text{-}1$$

$$f(i,k)=\frac{1}{NM}\sum_{m=0}^{M-1}\sum_{n=0}^{N-1} F(m,n)\,e^{j(2\pi/M)mi}\,e^{j(2\pi/N)nk}\qquad m=0,1,\cdots,M\text{-}1$$

$$n=0,1,\cdots,N\text{-}1$$



从计算公式中可以看出，DFT 的输入和输出均为离散值，非常适用于计算机的运算操作，因此利用计算机进行傅里叶变换的形式通常为 DFT。

根据可分离性，以上二维傅里叶变换可由连续两次一维傅里叶变换得到：

$$F(m,n)=\sum_{i=0}^{M-1}\left[\sum_{k=0}^{N-1}f(i,k)\mathrm{e}^{-j(2\pi/N)nk}\right]\mathrm{e}^{-j(2\pi/M)mi}$$

从而将二维 DFT 分解为水平和垂直两部分运算，上式中方括号中的项表示在图像的行上计算的 DFT，方括号外边的求和则实现结果数组在列上的 DFT，这种分解使我们可以用一维的 FFT 来快速实现二维 DFT，二维逆 DFT 也可以同样分解（离）。

6.1.3　快速傅里叶变换（FFT）

通常计算一维 DFT 所需的乘法和加法操作的次数是 N^2 次，因为它把所有的复指数值都存在一张表中，这样的计算量实在太大。而快速傅里叶算法将 DFT 计算式分解，可以将操作降到 $(N\log_2(N))$ 数量级，尤其当 N 是 2 的幂（即 $N=2^p$，其中 p 是整数）时，计算效率最高，实现起来也最简单。

其算法思想是，先将原图像进行转置，按行对转置后的图像矩阵做一维 FFT，将此变换所得的中间矩阵再转置，再按行对转置后的中间矩阵做一维 FFT，最后得到的就是二维 FFT。

6.1.4　傅里叶变换的应用

傅里叶变换在 MATLAB 的图像处理中具有广泛应用，例如进行滤波去噪、边缘检测等，在后续的章节中还将具体介绍，这里只举几个简单的例子。

1．快速卷积

傅里叶变换的性质表中的卷积定理指出了傅里叶变换的一个主要好处：与其在空域中做不直观的、难懂的卷积，不如在频域中做乘法，而且可以达到相同的效果。

根据卷积定理

$$F[\boldsymbol{A}\times\boldsymbol{B}]=F[\boldsymbol{A}]\cdot F[\boldsymbol{B}]$$

则有

$$\boldsymbol{A}\times\boldsymbol{B}=F^{-1}\{F[\boldsymbol{A}]\cdot F[\boldsymbol{B}]\}$$

下面的程序就是按上述原理实现快速卷积。首先构造两个矩阵 **A** 和 **B**，代码如下：

```
A = magic(3);    %用1到9之间的数产生一个3×3方阵
B = ones(3);     %生成一个全1的3×3方阵
```

```
A(8,8) = 0;        %用 0 将 A 补成 8×8 的方阵
B(8,8) = 0;        %用 0 将 B 补成 8×8 的方阵
```

然后，分别对 ***A*** 和 ***B*** 进行傅里叶变换，代码如下：

```
A2=fft2(A);
B2=fft2(B);
```

在频域内进行点乘，代码如下：

```
M= A2*B2;
```

最后将点乘的结果变换回空域，并截取有效数据，代码如下：

```
C = ifft2(M);
C = C(1:5,1:5);
C = real(C)
```

最后的结果为：

```
C =
   8.0000   9.0000 15.0000   7.0000    6.0000
  11.0000 17.0000 30.0000   19.0000   13.0000
  15.0000 30.0000 45.0000   30.0000   15.0000
   7.0000 21.0000 30.0000   23.0000    9.0000
   4.0000 13.0000 15.0000   11.0000    2.0000
```

2．模板匹配

模板匹配也是模式识别的一种有效方法。具体方法是：以待定位的目标为模板在待识别图像上滑动，同时做联贯运算，对运算结果取适当的阈值，确定待定位的目标的位置。联贯操作就是首先将模板图像旋转 180°，然后进行快速卷积。下面的图 6-16、图 6-17 就是采用所介绍的方法对图像中的字母“a”进行定位，得到的就是相关运算的结果。

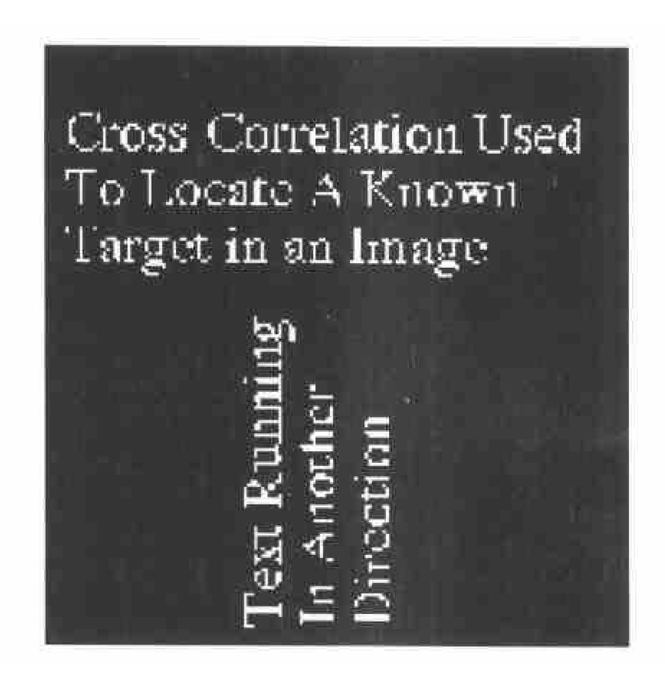
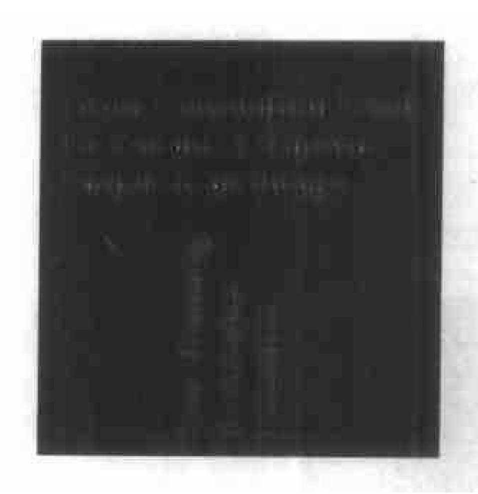


图 6-16　待识别图像和待匹配字符



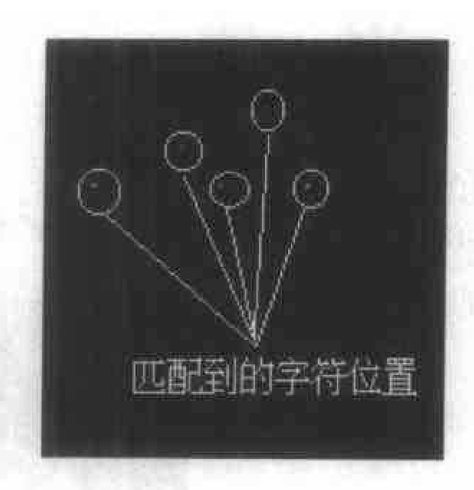


图 6-17 卷积的结果和文本匹配的结果

6.2 离散余弦变换及实现

离散余弦变换，简称 DCT，是一种实数域变换，其变换核为实数的余弦函数，计算速度较快，而且对于具有一阶马尔柯夫过程的随机信号，DCT 十分接近于 Karhunen-Loeve 变换，也就是说它是一种近似最佳变换，很适于做图像压缩和随机信号处理。

对于数字图像 $X(m,n)$，$0 \leqslant m \leqslant \mathrm{M}$，$0 \leqslant n \leqslant \mathrm{N}$，其二维 DCT 变换定义为

$$Y(k,l)=\alpha_k\alpha_l\sum_{m=0}^{\mathrm{M}-1}\sum_{n=0}^{\mathrm{N}-1}X(m,n)\cos\frac{(2m+1)k\pi}{2\mathrm{M}}\cos\frac{(2n+1)l\pi}{2\mathrm{N}}$$

$$k=0,1,\cdots,\mathrm{M}-1;\ l=0,1,\cdots,\mathrm{N}-1$$

其中，$\alpha_k=\begin{cases}1/\sqrt{\mathrm{M}}, & k=0\\ \sqrt{2/\mathrm{M}}, & 1\leqslant k\leqslant \mathrm{M}-1\end{cases}$，$\alpha_l=\begin{cases}1/\sqrt{\mathrm{N}}, & l=0\\ \sqrt{2/\mathrm{N}}, & 1\leqslant l\leqslant \mathrm{N}-1\end{cases}$

二维 DCT 变换具有可分离性，可以分解为双重的一维 DCT，实现起来非常方便。

二维 DCT 反变换（IDCT）定义为

$$X(m,n)=\alpha_k\alpha_l\sum_{m=0}^{\mathrm{M}-1}\sum_{n=0}^{\mathrm{N}-1}Y(k,l)\cos\frac{(2m+1)k\pi}{2\mathrm{M}}\cos\frac{(2n+1)l\pi}{2\mathrm{N}}$$

其中，$\alpha_k=\begin{cases}1/\sqrt{\mathrm{M}}, & k=0\\ \sqrt{2/\mathrm{M}}, & 1\leqslant k\leqslant \mathrm{M}-1\end{cases}$，$\alpha_l=\begin{cases}1/\sqrt{\mathrm{N}}, & l=0\\ \sqrt{2/\mathrm{N}}, & 1\leqslant l\leqslant \mathrm{N}-1\end{cases}$

由上式可知，原始图像 $X(m,n)$ 可表示为以 $Y(k,l)$ 为权值的一系列函数

$$\alpha_k\alpha_l\sum_{m=0}^{M-1}\sum_{n=0}^{N-1}\cos\frac{(2m+1)k\pi}{2M}\cos\frac{(2n+1)l\pi}{2N}\quad, \begin{matrix}0\leqslant k\leqslant M-1\\0\leqslant l\leqslant N-1\end{matrix}$$

的加权组合，这组函数就是 DCT 基函数。

图 6-18 是用图像方式显示8×8 DCT 基函数矩阵。

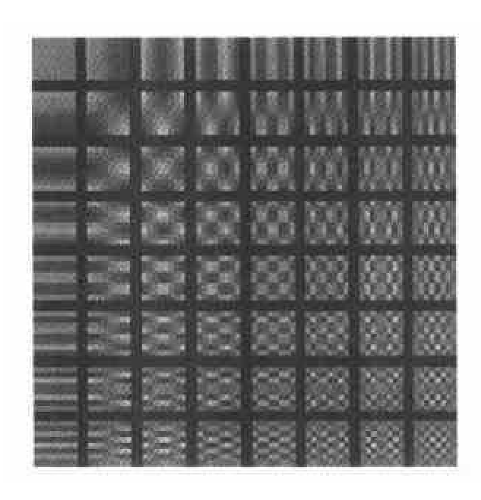


图 6-18　DCT 变换矩阵

DCT 变换的实现有两种方法，一种是基于 FFT 的快速算法，这是通过工具箱提供的 dct2 函数实现的；另一种是 DCT 变换矩阵（transform matrix）方法。变换矩阵方法非常适合做 8×8 或 16×16 的图像块的 DCT 变换，工具箱提供了 dctmtx 函数来计算变换矩阵。一个 M×M 的 DCT 变换矩阵 $\boldsymbol{T}$ 定义为

$$\boldsymbol{T}=(\boldsymbol{T}_{i,j})=\begin{cases}1/\sqrt{M} & i=0,\ 0\leqslant j\leqslant M-1\\ \sqrt{\dfrac{2}{M}}\cos\dfrac{\pi(2j+1)i}{2M} & 1\leqslant i\leqslant M-1,\ 0\leqslant j\leqslant M-1\end{cases}$$

则 $\boldsymbol{X}$ 的 DCT 变换 $\boldsymbol{Y}$ 为 $\boldsymbol{Y}=\boldsymbol{T}\boldsymbol{X}\boldsymbol{T}^T$。

离散余弦变换在图像压缩中有很多应用，它是 JPEG、MPEG 等数据压缩标准的重要数学基础。在 JPEG 压缩算法中，首先将 RGB 分量转化成亮度分量和色差分量，同时丢失一半的色彩信息（空间分辨率减半）；然后将输入图像划分为 8×8 或 16×16 的图像块，对每个图像块做 DCT 变换；然后舍弃高频的系数，并对余下的系数进行量化以进一步减少数据量；最后使用 RLE 和 Huffman 编码来完成压缩任务。解压缩时首先对每个图像块做 DCT 反变换，然后将图像块拼接成一幅完整的图像。下面是用 DCT 变换做图像压缩的例子。

例程 6-1 用 DCT 变换做图像压缩。

例程 6-1

```
I = imread('cameraman.tif');%读入图像
I = double(I)/255;
T = dctmtx(8);%计算离散变换矩阵，返回结果为双精度型
```

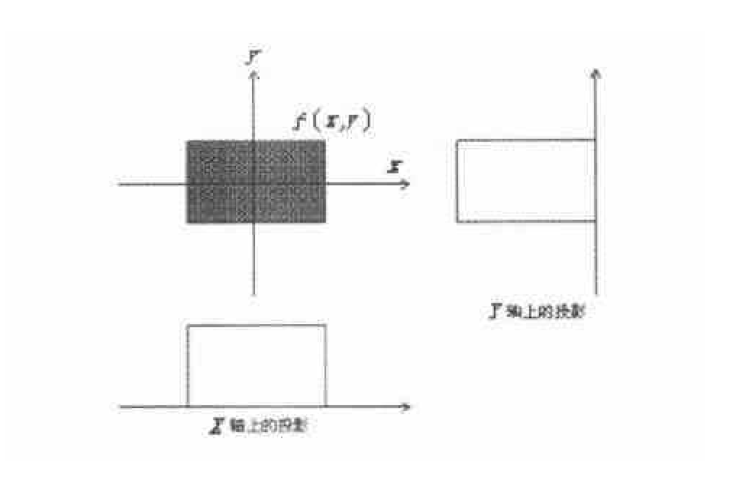


图 6-21　图像在坐标轴上的投影

图 6-22 是 Radon 变换的几何关系图。

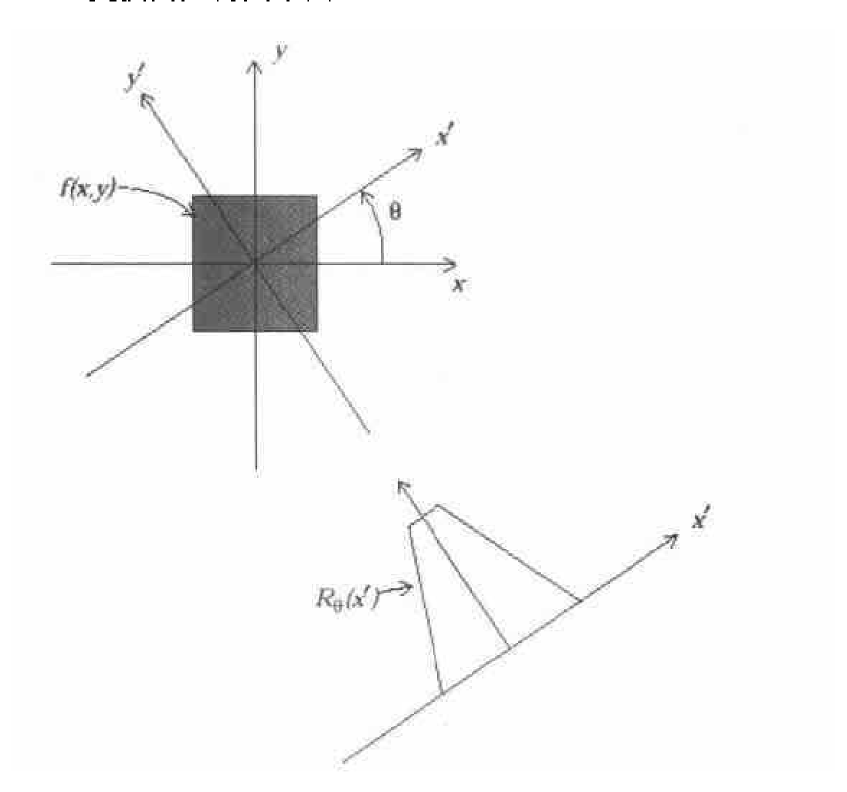


图 6-22　Radon 变换的几何关系

Radon 变换类似于计算机视觉中有名的 Hough 变换，可以用来检测图像中的直线，算法和程序如下。

例程 6-2 是 Radon 变换。原始图像见图 6-23。

例程 6-2

```
%首先，提取图像的局部边缘．见图 6-24
I = imread('ic.tif');
BW = edge(I);
```

```
imshow(I)
figure, imshow(BW)
%然后，对边缘图像做 Radon 变换，见图 6-25
theta = 0:179;
[R,xp] = radon(BW,theta);
imagesc(theta, xp, R);
colormap(hot);
xlabel('\theta (degrees)');
ylabel('x');
colorbar
```

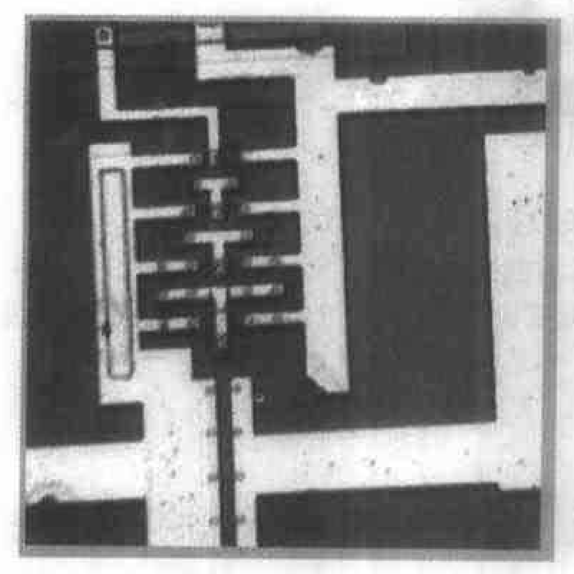


图 6-23　原始图像

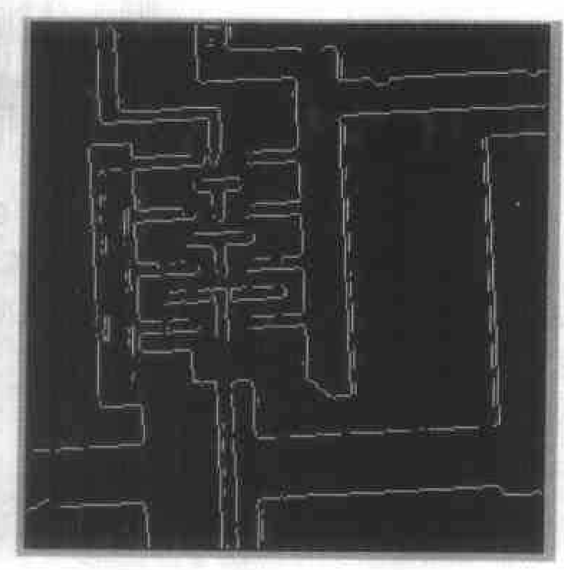


图 6-24　提取图像的局部边缘

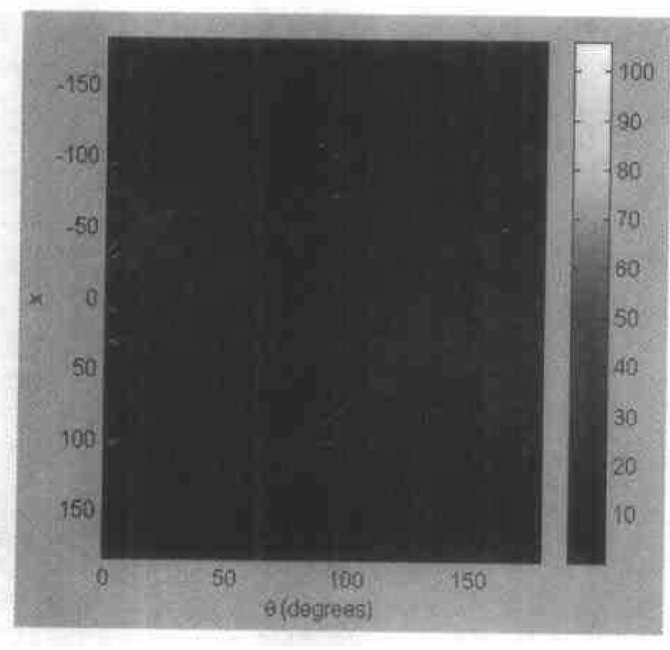


图 6-25　边缘图像的 Radon 变换

最后，找出变换矩阵 **R** 的峰值，这些峰值对应着原图像上的直线。比如，Radon 变换矩阵在 $x' = -101$、$\theta = 94°$ 处有一个峰值，则对应原图在倾角 $\theta = 94°$ 的坐标轴 X' 的-101 处有一条垂直于 X' 的直线，见图 6-26。

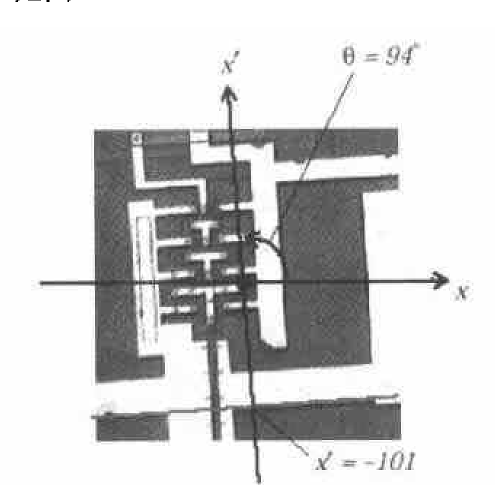


图 6-26　利用 Radon 变换提取的一条直线段

radon 函数的反变换通常应用于立体图像的剖面图分析。因为与 radon 变换相反，所以 radon 函数的反变换可以用来利用投影数据重构原始图像。Radon 函数是从平行光束投影中重构原始图像。

第 7 章　FIR 滤波器设计

FIR 滤波器，就是有限脉冲响应滤波器，它具有以下优点：

- 易于表征为系数矩阵。
- 二维 FIR 滤波器是一维 FIR 滤波器的自然扩展
- 有多种成熟的滤波器设计方法。
- 易于实现。
- 能够设计严格的线性相位，而同时可以具有任意幅度特性，从而避免了图像处理中的变形。
- FIR 滤波器的单位抽样响应是有限长的，因而是稳定的。

对于图像处理来说，最常用的滤波器是二维 FIR 滤波器。

本章主要内容：

- FIR 滤波器设计的窗口方法
- FIR 滤波器设计的频率采样方法
- FIR 滤波器设计的频率变换方法

7.1　FIR 滤波器设计基础

FIR 滤波器的设计通常是在频域内进行的，为了分析 FIR 滤波器的频域特性，图像处理工具箱提供了计算二维线性系统频率响应的函数 freqz2 和计算期望频率响应矩阵的 freqspace 函数。

1．freqz2 函数

freqz2 函数语法如下：

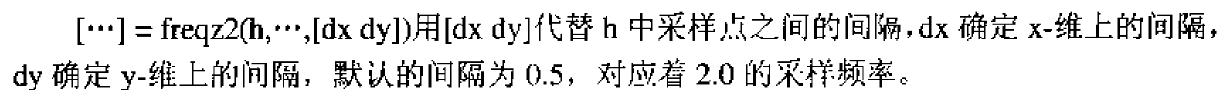
```
[H,f1,f2] = freqz2(h,n1,n2)
[H,f1,f2] = freqz2(h,[n2 n1])
[H,f1,f2] = freqz2(h,f1,f2)
[H,f1,f2] = freqz2(h)
[…] = freqz2(h,…,[dx dy])
[…] = freqz2(h,…,dx)
freqz2(…)
```

[H,f1,f2] = freqz2(h,n1,n2) 返回 FIR 滤波器 h 的 n1×n2 点频率响应矩阵 H，n1、n2 的默认值为 64。f1 和 f2 为对应的频率轴矢量，取归一化频率[-1，1]，其中 1 对应 Nequist 频率（即采样率的一半）。h 以二维 FIR 滤波器的计算核的形式出现。

[H,f1,f2] = freqz2(h,[n2 n1]) 与[H,f1,f2] = freqz2(h,n1,n2)返回的结果一样。

[H,f1,f2] = freqz2(h,f1,f2) 返回指定频率坐标（f1,f2）处的频率响应 H。

[H,f1,f2] = freqz2(h)是[n2 n1] = [64 64]时[H,f1,f2] = freqz2(h,[n2 n1])的特殊情况。



[…] = freqz2(h,…,[dx dy])用[dx dy]代替 h 中采样点之间的间隔，dx 确定 x-维上的间隔，dy 确定 y-维上的间隔，默认的间隔为 0.5，对应着 2.0 的采样频率。

[…] = freqz2(h,…,dx)用 dx 来确定两个坐标上的采样间隔。

freqz2(…) 无返回值，只生成二维幅频响应的网格图。

数据类型要求：输入矩阵可以是 double 型或任何整数类型的，所有其他的输入都必须是 double 型，所有的输出都是 double 型的。

例程 7-1 已知二维滤波器 h 的计算核，计算并显示 h 的 64×64 点的频率响应。

例程 7-1

```
h=[0.1667 0.6667 0.1667;0.6667 -3.3333 0.6667;0.1667 0.6667 0.1667]
h =
0.1667     0.6667     0.1667
0.6667    -3.3333     0.6667
0.1667     0.6667     0.1667
%计算并显示 h 的 64×64 点的频率响应
freqz2(h)
```

结果如图 7-1 所示。

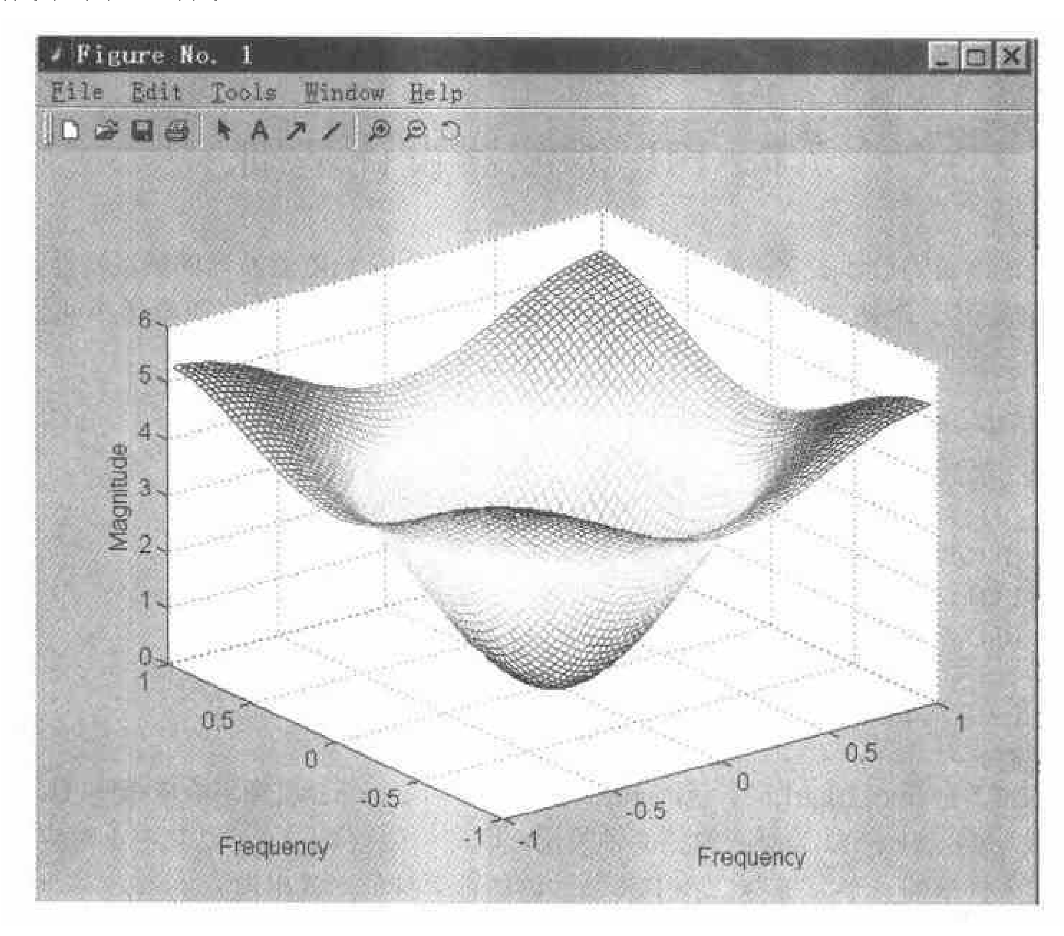


图 7-1 二维 FIR 滤波器的幅频响应

2．freqspace 函数

freqspace 函数返回等间隔频率响应的隐含频率范围，因为滤波器设计函数 fsample2、fwind1 和 fwind2 都是基于期望频率响应矩阵来设计过滤器的，所以通常都要先调用 freqspace 函数。

freqspace 函数的语法格式如下：

```
[f1,f2] = freqspace(n)
[f1,f2] = freqspace([m n])
[x1,y1] = freqspace(…,'meshgrid')
f = freqspace(N)
f = freqspace(N,'whole')
```

[f1,f2] = freqspace(n)对应一个 n×n 矩阵，返回二维频率矢量 f1 和 f2。当 n 为奇数时，f1 和 f2 都是[−n+1：2：n−1]/n；当 n 为偶数时，f1 和 f2 都是 [−n：2：n−2]/n。

[f1,f2] = freqspace([m n]) 对一个 m×n 矩阵，返回二维频率矢量 f1 和 f2。

[x1,y1] = freqspace(…,'meshgrid') 等价于[f1,f2] = freqspace(…)。

[x1,y1] = meshgrid(f1,f2)。

f = freqspace(N) 假设 N 个均匀分布在单位圆上的点，返回一维频率矢量 f。无论 N 是奇数或偶数，f 都是(0：2/N：1)。当 N 是奇数时，返回(N+1)/2 个点；当 N 是偶数时，返回(N+2)/2 个点。

f = freqspace(N,'whole')返回 N 个均匀分布在整个单位圆上的点，在这种情况下，f 是 0：2/N：2×(N−1)/N。

例程 7-2 产生一个圆形的理想低通频率响应。

例程 7-2

```
[f1,f2]=freqspace(25,'meshgrid');
Hd=zeros(25,25);
d=sqrt(f1.^2+f2.^2)<0.5;
Hd(d)=1;
mesh(f1,f2,Hd)
```

得到的期望的频率响应矩阵如图 7-2 所示。

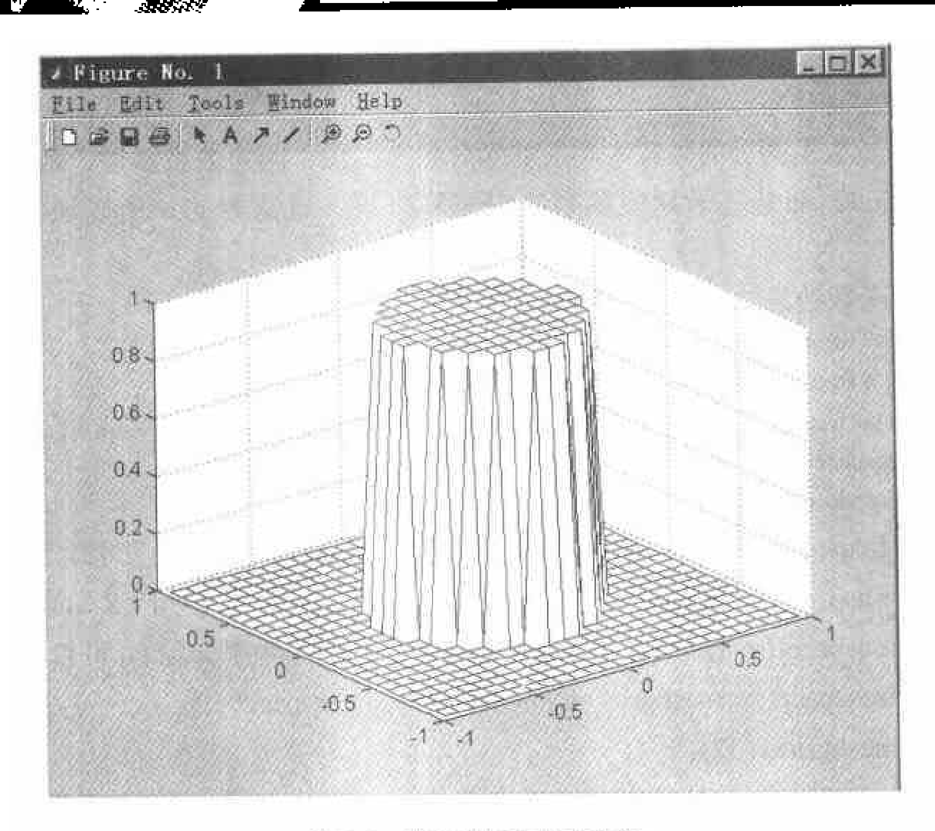


图 7-2 期望的频率响应矩阵

7.2 窗 口 方 法

二维窗函数设计法是二维滤波器设计的最早技术之一，它是一维窗函数方法的直接推广。它根据所要求的理想滤波器频率响应，设计一个 FIR 滤波器频率响应来逼近理想滤波器频率响应。

设要设计的二维 FIR 滤波器的单位脉冲响应为

$$h(n_1,n_2)=\omega(n_1,n_2)\cdot i(n_1,n_2)$$

其中 $i(n_1,\ n_2)$是无限冲激响应序列，$\omega(n_1,\ n_2)$是二维窗函数。等式两边做傅里叶变换，得

$$H(\omega_1,\omega_2)=\frac{1}{4\pi^2}\int\int I(\Omega_1,\Omega_2)W(\omega_1-\Omega_1,\omega_1-\Omega_2)\mathrm{d}\Omega_1\,\mathrm{d}\Omega_2$$

上式表明，对无限冲激响应加窗的效果等价于在频域用 $W(\omega_1,\omega_2)$对 $I(\omega_1,\omega_2)$进行平滑。

常用的窗函数有矩形（boxcar）窗、Bartlett 窗（三角窗）、海明（Hamming）窗、汉宁（Hanning）窗、布莱克曼（Blackman）窗和凯泽（Kaiser）窗等。

窗口方法有两种：一种方法是用一维窗函数来构造二维窗函数，也就是使用函数 fwind1，将两个一维窗函数直接相乘，得到二维窗函数，或者将一维窗函数旋转得到二维窗函数；另一种方法是使用二维窗函数，也就是使用函数 fwind2。

1．fwind1 函数

fwind1 函数的语法格式如下：

```
h = fwind1(Hd,win)
h = fwind1(Hd,win1,win2)
h = fwind1(f1,f2,Hd,…)
```

h = fwind1(Hd,win) 由期望的幅频响应 Hd，使用黄氏算法，从一维窗函数 win 得到近似圆对称的二维窗函数。矩阵 Hd 中的元素是在 X、Y 频率轴的[-1.0,1.0]（在频率归一化中，1.0 对应着采样频率的一半，或π）区间内等间隔采样得到的点。为了得到精确的结果，可以使用 freqspace 函数返回的频率点来生成 Hd。窗函数 win 可以取以下值：boxcar、Hamming、Hanning、Bartlett、Blackman、Kaiser 或 Chebwin，如果 win 的长度是 n，那么 h 是 $n \times n$ 的。返回值 h 是计算核的形式，可以直接应用于 filter2 函数中。

h = fwind1(Hd,win1,win2) 用两个一维窗函数 win1 和 win2 相乘生成可分离的二维窗。如果 win1 的长度是 n，win2 的长度是 m，那么 h 是 $m \times n$ 的。

h = fwind1(f1,f2,Hd,…) 指定了任意频率坐标(f1,f2)处的频率响应 Hd，f1 和 f2 都采用归一化频率。窗的长度控制着所得滤波器的尺寸

数据类型要求：输入矩阵可以是 double 型的，或任何整数类型的。所有其他的输入必须是 double 型的。所有的输出都是 double 型的。

黄氏算法：

- 用一维窗函数生成近似圆对称的二维窗函数

$$w(n_1,n_2) = w(t)\big|_{t=\sqrt{n_1^2+n_2^2}}$$

其中 $w(t)$ 是一维窗函数，$w(n_1,n_2)$ 是得到的二维窗函数。

- 给定两个一维窗函数，生成可分离的二维窗函数

$$w(n_1,n_2) = w_1(n_1)w_2(n_2)$$

fwind1 以 Hd 和二维窗函数为参数调用 fwind2，fwind2 先对 Hd 进行逆傅里叶变换，并与二维窗函数相乘，得到 h。

$$h_d(n_1,n_2) = \frac{1}{(2\pi)^2}\int_{-\pi}^{\pi}\int_{-\pi}^{\pi} H_d(w_1,w_2)\,\mathrm{e}^{jw_1n_1}\,\mathrm{e}^{jw_2n_2}\,\mathrm{d}w_1\,\mathrm{d}w_2$$

$$h(n_1,n_2) = h_d(n_1,n_2)w(n_1,n_2)$$

例程 7-3 设计带通滤波器。

例程 7-3

```
%分别采用 Hamming 窗、Bartlett 窗、Hanning 窗和 Blackman 窗设计近似圆对称的
%带通滤波器，通频带为[0.1,0.5]。
%首先，创建包含了期望的带通响应的矩阵 Hd
```

```
[f1,f2] = freqspace(21,'meshgrid');
Hd = ones(21);
r = sqrt(f1.^2 + f2.^2);
Hd((r<0.1)|(r>0.5)) = 0;
colormap(jet(64))
mesh(f1,f2,Hd)
%期望的频率响应矩阵 Hd 如图 7-3 所示
```

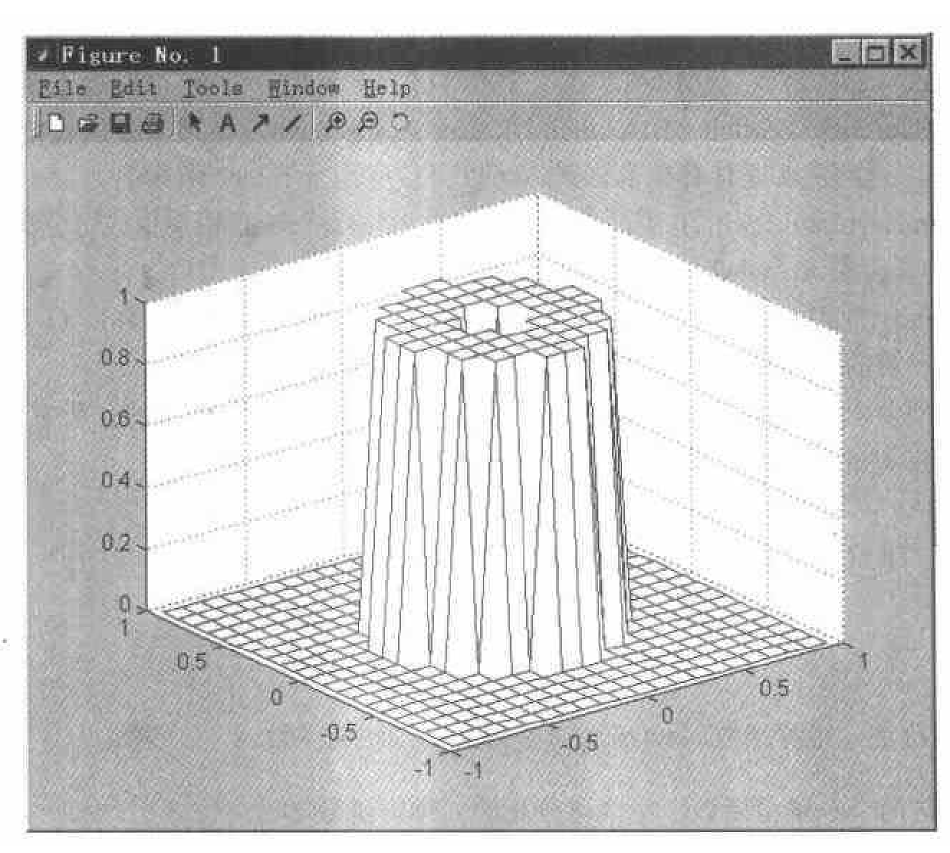


图 7-3　期望的频率响应矩阵 Hd

```
%然后，分别用一维 Hamming 窗、Bartlett 窗、Hanning 窗和 Blackman 窗设计滤波器，
%代码如下：
h1 = fwind1(Hd,hamming(21));
figure,freqz2(h1)
h2 = fwind1(Hd, bartlett (21));
figure,freqz2(h2)
h3 = fwind1(Hd, hanning (21));
figure,freqz2(h3)
h4 = fwind1(Hd, blackman (21));
figure,freqz2(h4)
```

四个滤波器的频率响应分别如图 7-4、图 7-5、图 7-6 和图 7-7 所示。

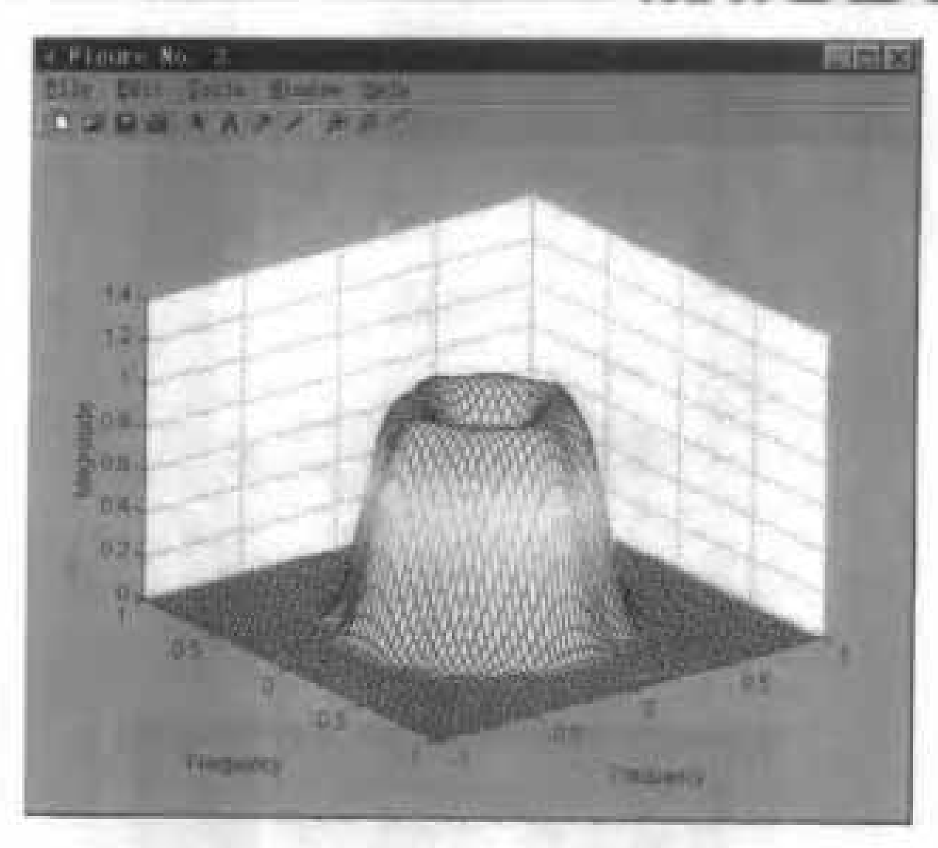


图 7-4 用 Hamming 窗设计的带通滤波器幅频响应

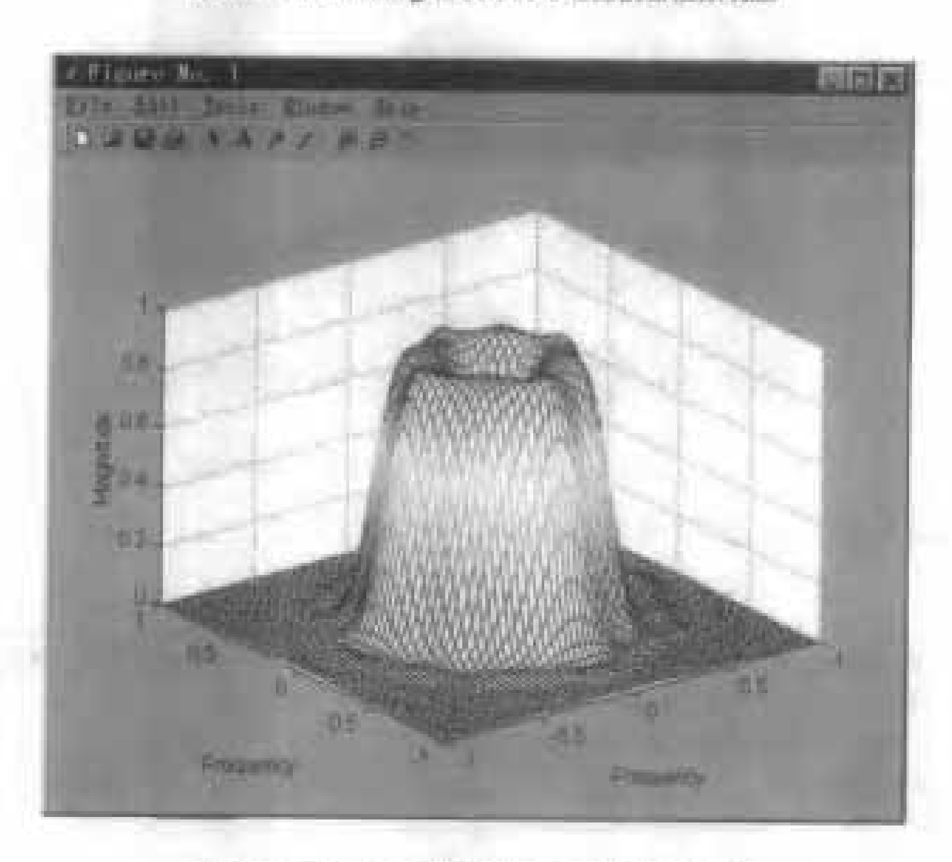


图 7-5 用 Bartlett 窗设计的带通滤波器幅频响应

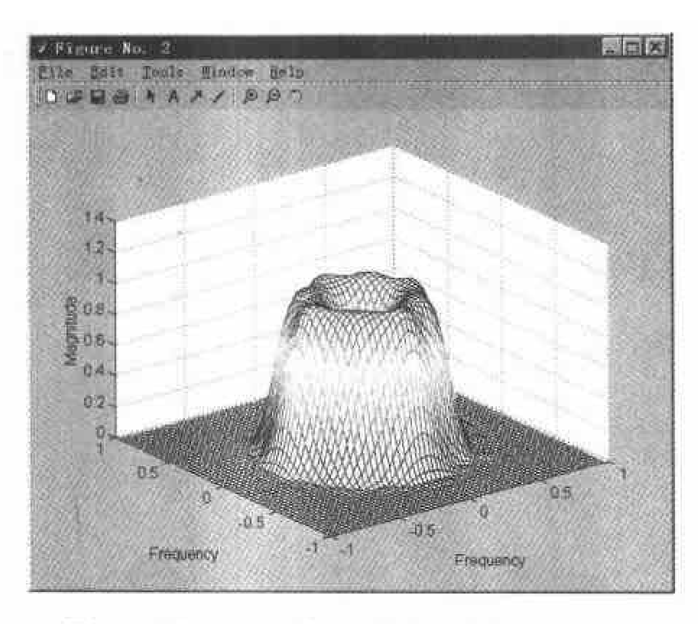


图 7-6 用 Hanning 窗设计的带通滤波器幅频响应

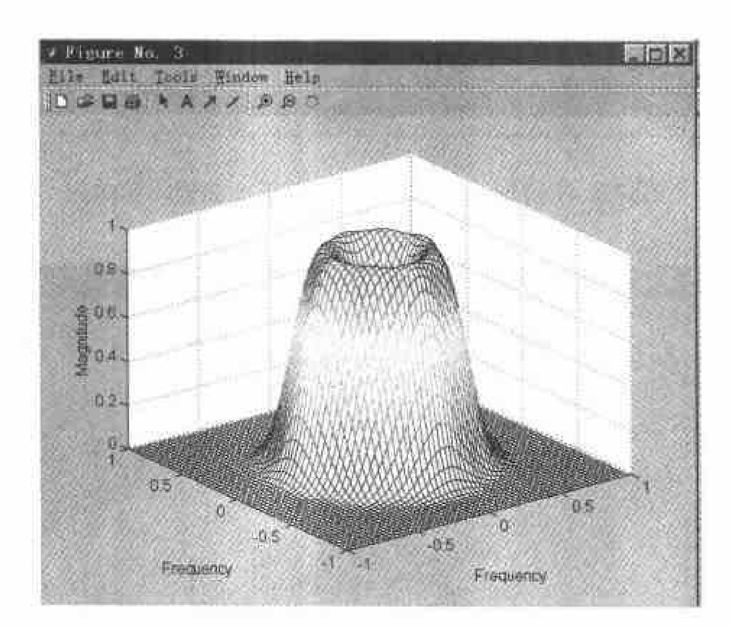


图 7-7 用 Blackman 窗设计的带通滤波器幅频响应

例程 7-4 使用上述 Hd，用两个一维窗口设计二维滤波器。

例程 7-4

```
%第一个滤波器用一维的 Hamming 窗和一维的 Bartlett 窗，第二个滤波器用一维的
%Hanning 窗和一维的 Blackman 窗
h5= fwind1(Hd,hamming(21), bartlett(21));
figure,freqz2(h5)
h6 = fwind1(Hd,hanning(21),blackman(21));
figure,freqz2(h6)
```

得到的带通滤波器分别如图 7-8 和 7-9 所示。

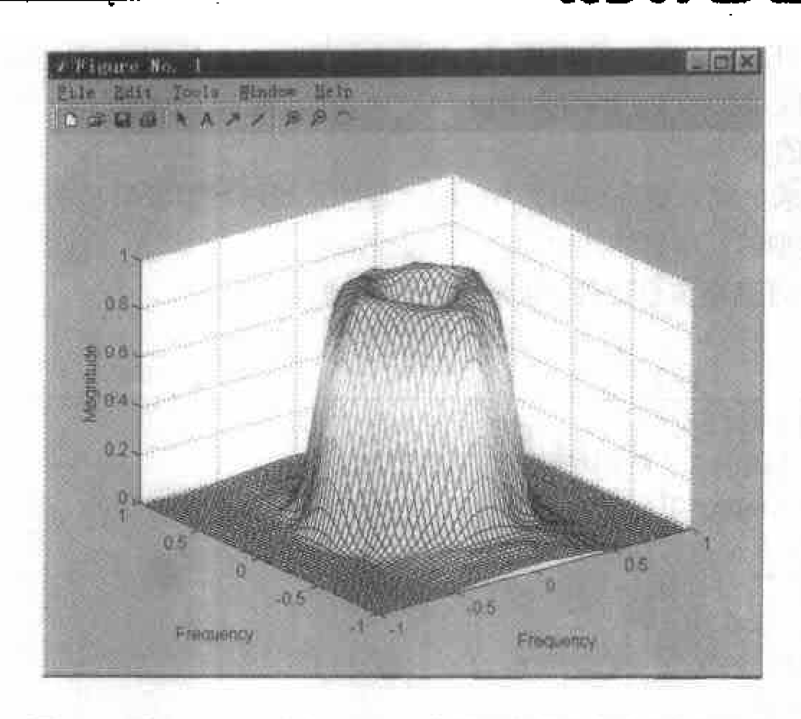


图 7-8　用 Hamming 窗和 Bartlett 窗设计的带通滤波器幅频响应

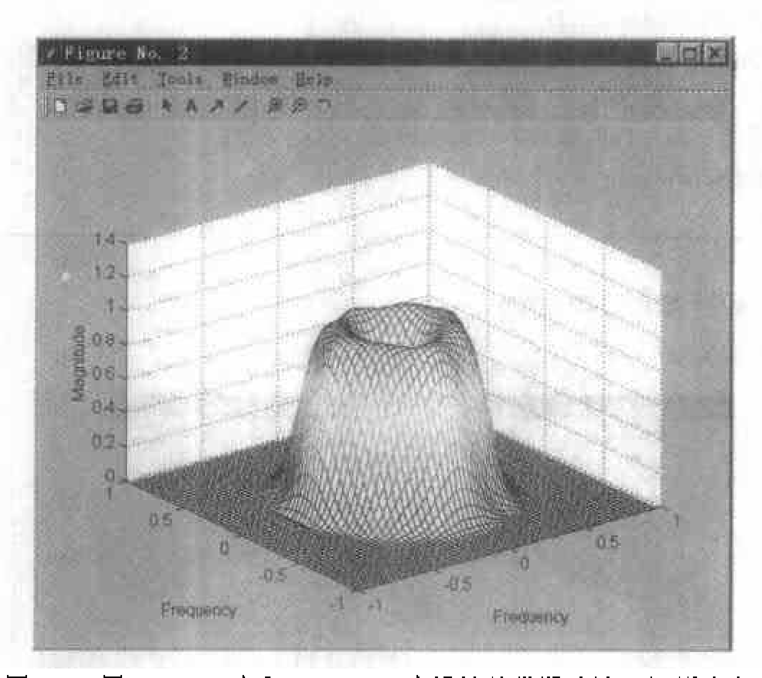


图 7-9　用 Hanning 窗和 Blackman 窗设计的带通滤波器幅频响应

2．fwind2 函数

fwind2 函数的语法格式如下：

```
h = fwind2(Hd,win)
h = fwind2(f1,f2,Hd,win)
```

h = fwind2(Hd,win)生成二维 FIR 滤波器的方法是：对期望频率响应 Hd 进行逆傅里叶变换，然后与窗函数 win 相乘。矩阵 Hd 包含在笛卡儿平面上等间隔点处的期望频率响应。返回值 h 是计算核的形式，可以直接用于 filter2 函数中，h 的尺寸等于 win 的尺寸。为了得到精确的结果，可以使用 freqspace 函数返回的频率点来生成 Hd。

h = fwind2(f1,f2,Hd,win) 指定了 *X*、*Y* 轴上任意频率坐标(f1,f2)处的幅频响应 Hd。频率矢量 f1 和 f2 应该在[-1.0,1.0]的范围内，其中 1.0 对应着采样频率的一半，或π弧度。h 的尺寸等于 win 的尺寸。

数据类型要求：输入矩阵可以是 double 型的，或任何整数类型的。所有其他的输入都必须是 double 型的。所有的输出都是 double 型的。

例程7-5用二维高斯窗设计近似圆对称的带通滤波器。

例程 7-5

```
%通频带为[0.1,0.5]
%首先，创建包含了期望的带通响应的矩阵 Hd
[f1,f2] = freqspace(21,'meshgrid');
Hd = ones(21);
r = sqrt(f1.^2 + f2.^2);
Hd((r<0.1)|(r>0.5)) = 0;
colormap(jet(64))
mesh(f1,f2,Hd)
%然后，生成旋转对称的二维高斯窗函数
win = fspecial('gaussian',21,2);
win = win ./ max(win(:));  % 将窗口的最大值归 1
mesh(win)
%最后，用前一步生成的窗函数设计滤波器，代码如下：
h1 = fwind2(Hd,win);
freqz2(h1)
```

期望频率响应矩阵如图 7-10 所示。

二维高斯窗函数如图 7-11 所示。

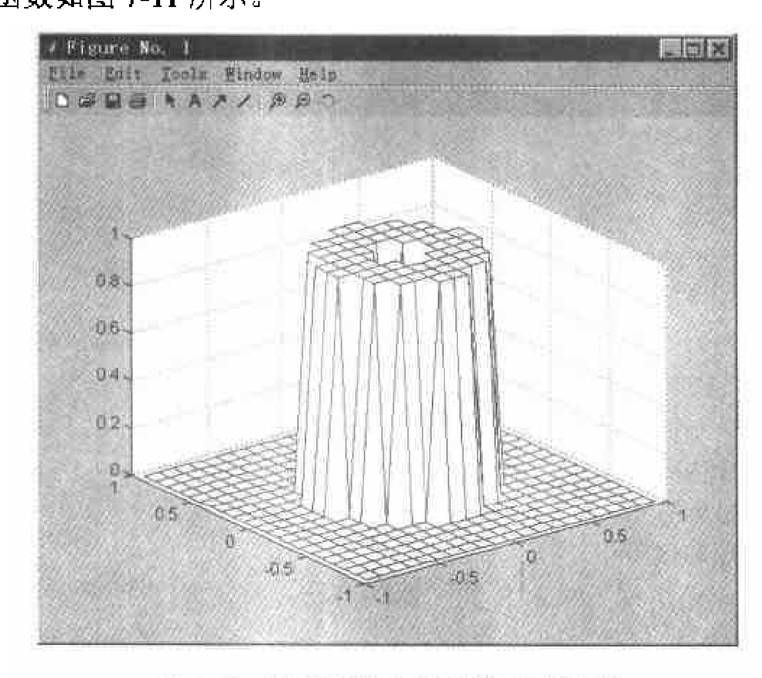


图 7-10　期望频率响应矩阵的幅频响应

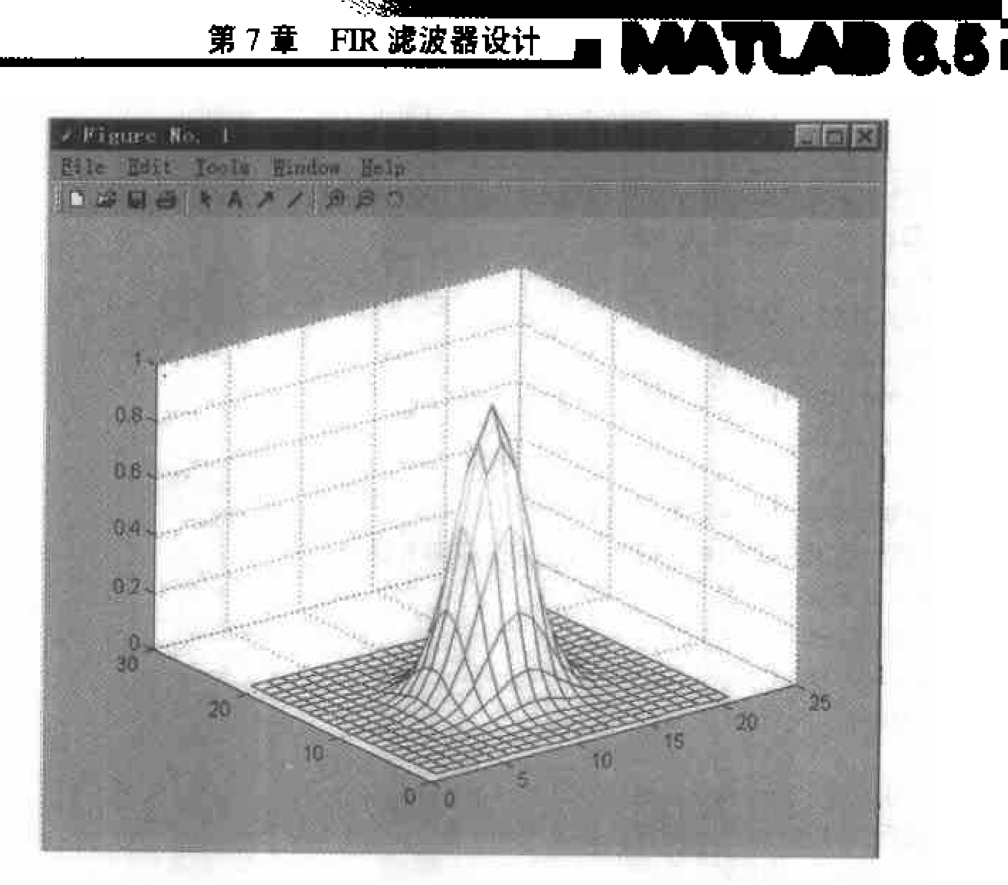


图 7-11 二维高斯窗函数

二维窗函数法设计的带通滤波器的频率响应如图 7-12 所示。

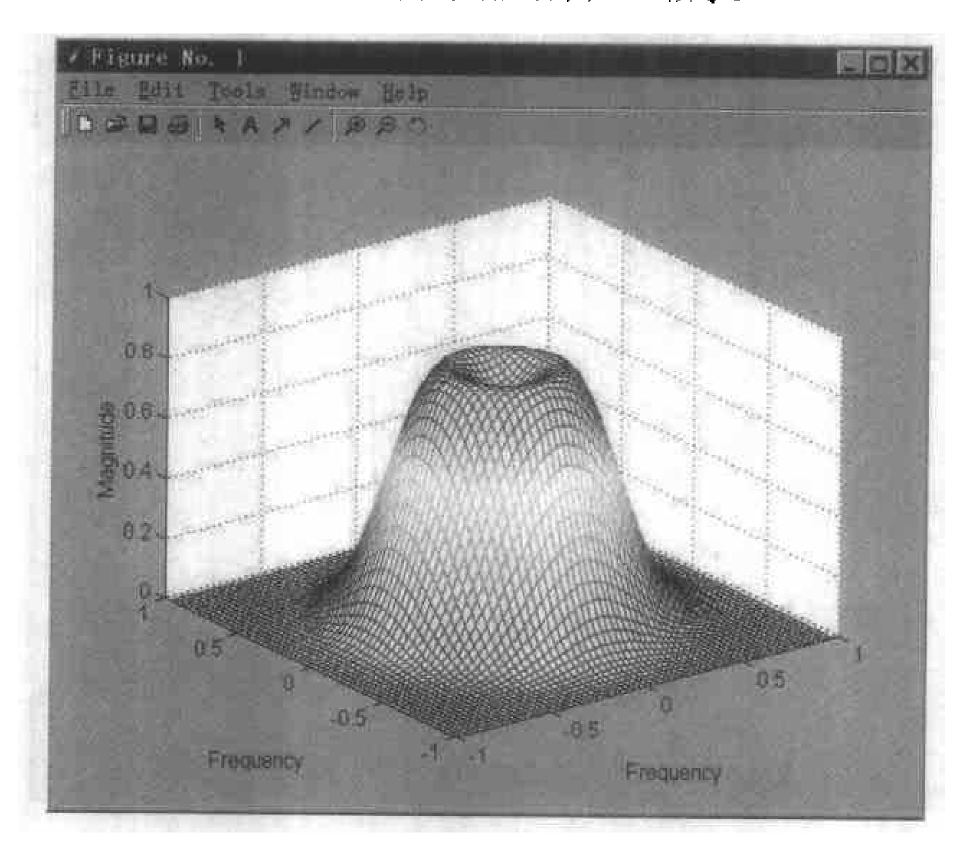


图 7-12 用二维窗函数法设计的低通滤波器的频率响应

例程 7-6 用二维拉普拉斯—高斯窗设计近似圆对称的高通滤波器。

例程 7-6

```
%首先，创建包含了期望的带通响应的矩阵 Hd。代码如下：
[f1,f2] = freqspace(21,'meshgrid');
Hd = ones(21);
r = sqrt(f1.^2 + f2.^2);
Hd(r<0.5) = 0;
colormap(jet(64))
mesh(f1,f2,Hd)

%期望频率响应矩阵如图 7-13 所示
%然后，生成旋转对称的二维拉普拉斯—高斯窗函数
win = fspecial('log',21,2);
% 将窗口的最大值归 1
win = win ./ max(win(:));
mesh(win)

% 二维拉普拉斯—高斯窗函数如图 7-14 所示
%最后，用前一步生成的窗函数设计滤波器
h2= fwind2(Hd,win);
freqz2(h2)
```

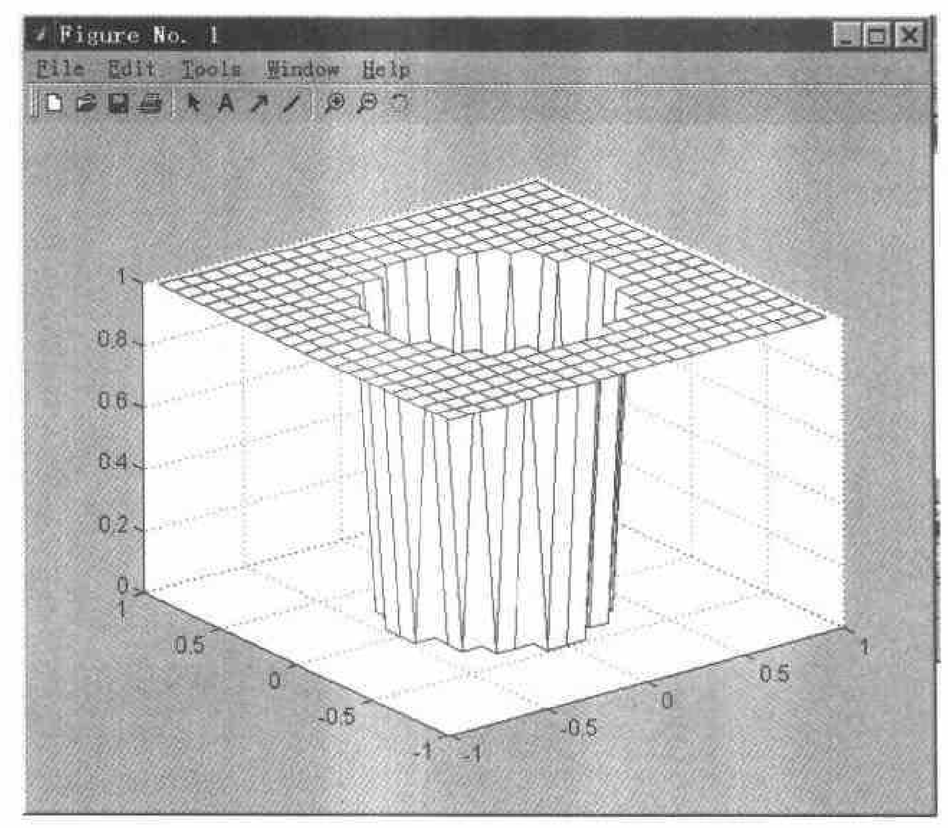


图 7-13　期望频率响应矩阵的频率响应

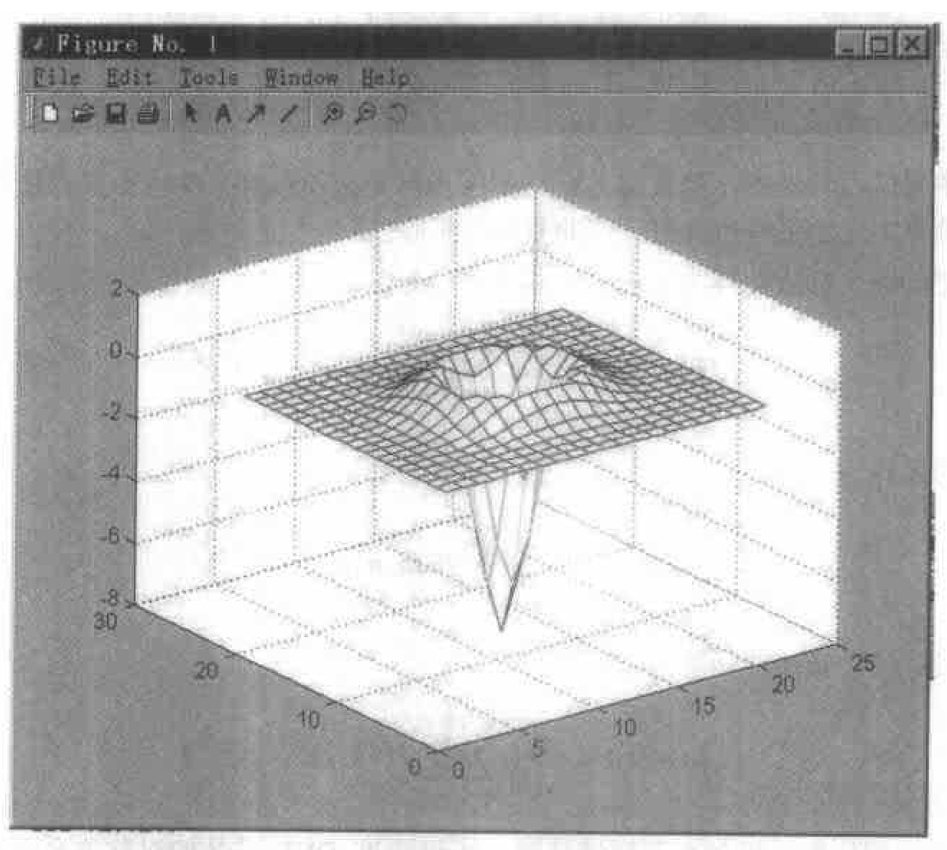


图 7-14　二维拉普拉斯—高斯窗函数

二维窗函数法设计的高通滤波器的频率响应如图 7-15 所示。

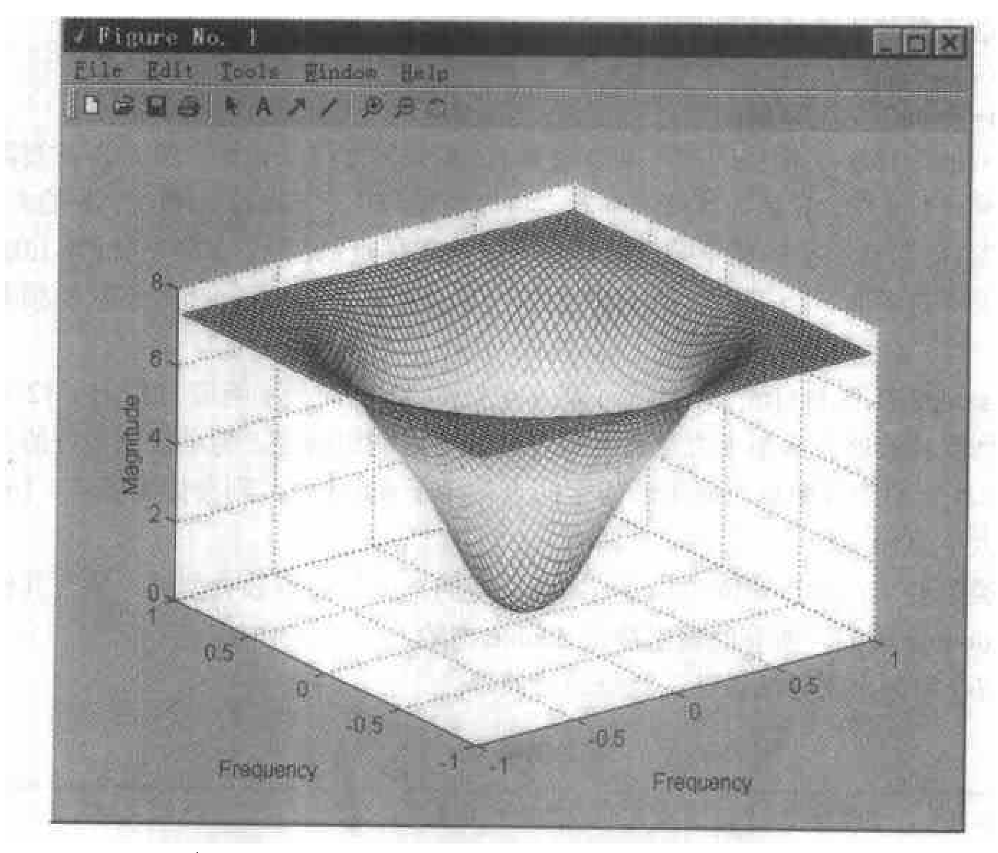


图 7-15　二维窗函数法设计的高通滤波器的频率响应

7.3 频率采样法

在频率采样设计法中，滤波器的频率响应指标是由连续频率响应的采样值得到的，用其中非零频率响应的采样值就可计算冲激响应矩阵。

设期望的频率响应矩阵为 $h(m,n), 0\leqslant m\leqslant \mathrm{M}, 0\leqslant n\leqslant \mathrm{N}$，其频率响应为

$$H(\mathrm{e}^{j\omega_1},\mathrm{e}^{j\omega_2})=\sum_{m=0}^{\mathrm{M}-1}\sum_{n=0}^{\mathrm{N}-1}h(m,n)\mathrm{e}^{-j(\omega_1 m+\omega_2 n)}$$

频域的采样为

$$H(k,l)=\sum_{m=0}^{\mathrm{M}-1}\sum_{n=0}^{\mathrm{N}-1}h(m,n)\mathrm{e}^{-j2\pi(\frac{mk}{\mathrm{M}}+\frac{nl}{\mathrm{N}})}$$

则所设计的冲激响应矩阵为

$$h(m,n)=\frac{1}{\mathrm{MN}}\sum_{k=0}^{\mathrm{M}-1}\sum_{l=0}^{\mathrm{N}-1}H(k,l)\mathrm{e}^{j2\pi(\frac{mk}{\mathrm{M}}+\frac{nl}{\mathrm{N}})}$$

对于线性相位的二维频率采样滤波器，还可通过 DFT 系数的对称关系进一步简化计算过程。

图像处理工具箱中用频率采样法设计二维滤波器的函数是 fsamp2。

fsamp2 函数的语法格式如下：

```
h = fsamp2(Hd)
h = fsamp2(f1,f2,Hd,[m n])
```

h = fsamp2(Hd) 由期望的频率响应矩阵 Hd 用频域采样法设计二维 FIR 滤波器，滤波器的系数返回到 h 中，h 是计算核的形式，可以直接应用于 filter2 函数中。h 的大小与 Hd 相同。矩阵 Hd 包含期望的频率响应采样点，这些采样点是在 *X*、*Y* 频率轴的[-1.0,1.0]区间上等间隔采样得到的。为了得到精确的结果，可以使用 freqspace 函数返回的频率点生成 Hd。

h = fsamp2(f1,f2,Hd,[m n]) 指定了频率坐标(f1,f2)处的频率响应 Hd，f1、f2 采用相对频率。得到的滤波器在最小平方意义上尽可能地逼近期望响应。为了得到最好的结果，至少要有 m×n 个期望频率点，如果所指定的点数少于 m×n 个，则会提出警告。[m n]指定了 h 矩阵大小为 m×n 。

数据类型要求：输入矩阵可以是 double 型的，或任何整数类型的。所有其他的输入都必须是 double 型的。所有的输出都是 double 型的。

例程 7-7 用频率采样法设计二维低通滤波器。

例程 7-7

```
%首先，创建包含了期望的频率响应的矩阵 Hd
[f1,f2] = freqspace(21,'meshgrid');
Hd = zeros(21);
```

```
Hd(7:15,7:15) = 1;
Axis([-1 1 -1 1 0 1.2]),
colormap(jet(64))
mesh(f1,f2,Hd)

%期望的滤波器的频率响应见图 7-16
%然后，设计通过此响应的滤波器
h = fsamp2(Hd);
figure,freqz2(h)
```

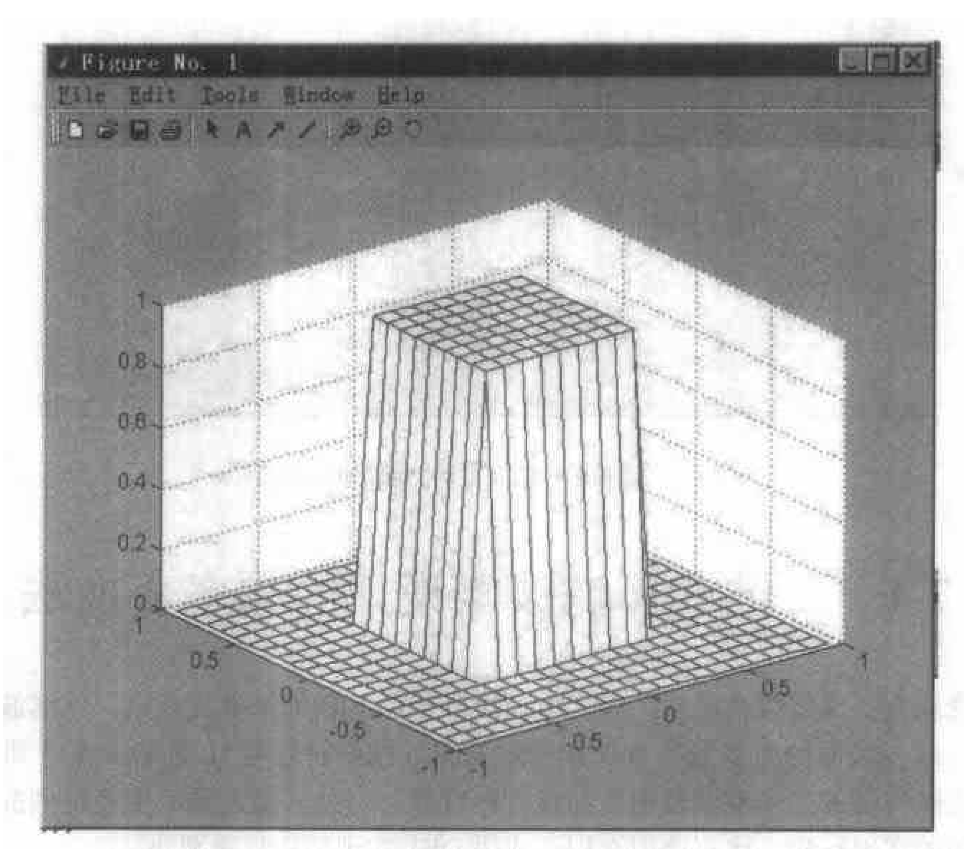


图 7-16　期望的滤波器频率响应图

用频域采样法设计的滤波器的频率响应见图7-17。比较期望频率响应与实际的滤波器的频率响应可以看到明显的波纹，这些波纹是频率采样设计方法中的重要问题，出现在期望的响应中较明显的过渡中。读者可以通过使用较大的滤波器来降低波纹的空间区域，但并不能减低波纹的高度，而且需要更多的滤波计算时间。为了得到期望频率响应的更加光滑的近似值，可以考虑使用频率变换方法或窗口方法。

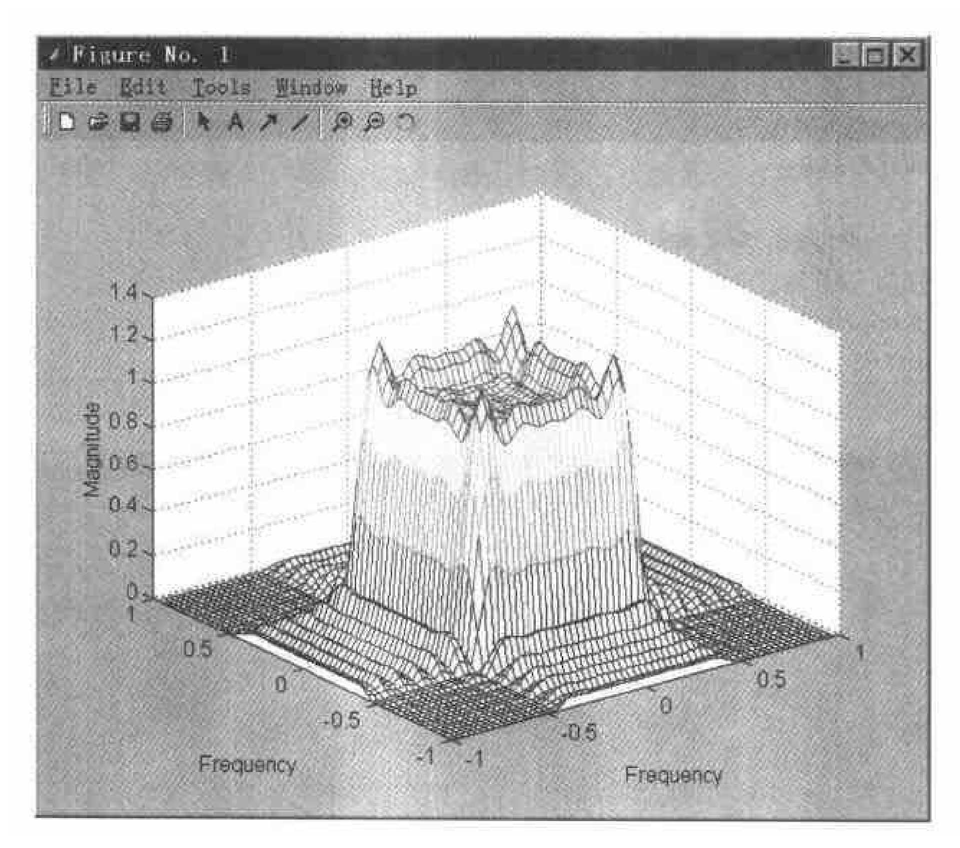


图 7-17　频域采样法设计的滤波器的频率响应

7.4　二维 FIR 滤波器设计的频率变换法

频率变换方法通过频率变换，将一个一维零相位 FIR 滤波器变换成二维滤波器。由于它保持了一维滤波器的大多数特性（如传输带宽、纹波特性等），所以如果一维滤波器是最优的，变换得到的二维滤波器也容易保持最优性。另外，通过频率变换得到的二维滤波器具有有效的实现结构，比直接用卷积法或用 DFT 实现的运算量要小。

MATLAB 图像处理工具箱提供了函数 ftrans2，用于实现频率变换算法。

ftrans2函数的语法格式如下：

```
h = ftrans2(b,t)
h = ftrans2(b)
```

h = ftrans2(b,t)由一个一维 FIR 滤波器的冲激响应 b 通过频率变换 t 生成二维滤波器 h。t 为变换矩阵，默认地采用 McClellan 变换矩阵，即

$$t=\frac{1}{8}\begin{bmatrix}1 & 2 & 1\\ 2 & -4 & 2\\ 1 & 2 & 1\end{bmatrix}$$

b 必须是一维的长度为奇数（第一类）的 FIR 滤波器。例如，可以由 fir1 函数、fir2 函数或 remez 函数得到。返回值 h 是计算核的形式，可以直接用于 filter2 函数。如果 t 是 $m\times n$ 维的，b 的长度是 Q，那么 h 的尺寸是

$$\left(\frac{(m-1)\times(Q-1)}{2}+1\right)\times\left(\frac{(n-1)\times(Q-1)}{2}+1\right)$$

h = ftrans2(b) 使用默认的 McClellan 变换矩阵。

ftrans2 返回的二维滤波器的频率响应由以下变换定义：

$$H(w_1,w_2)=B(w)\Big|_{\cos w=T(w_1,w_2)}$$

其中 $B(w)$ 是一维滤波器 b 的傅里叶变换：

$$B(w)=\sum_{n=-N}^{N}b(n)\mathrm{e}^{-jwn}$$

而 $T(w_1,w_2)$ 是变换矩阵 t 的傅里叶变换：

$$T(w,w)=\sum_{n_2}\sum_{n_1}t(n_1,n_2)\mathrm{e}^{-jw_1n_1}\mathrm{e}^{-jw_2n_2}$$

最后返回的滤波器 **h** 是 $H(w_1,w_2)$ 逆傅里叶变换：

$$h(n_1,n_2)=\frac{1}{(2\pi)^2}\int_{-\pi}^{\pi}\int_{-\pi}^{\pi}H(w_1,w_2)\mathrm{e}^{jw_1n_1}\mathrm{e}^{jw_2n_2}\,\mathrm{d}w_1\,\mathrm{d}w_2$$

例程 7-8 用频率变换法设计近似圆对称的二维 FIR 低通滤波器。

例程 7-8

```
%首先，创建一个一维FIR低通滤波器，变换矩阵是可以根据需要定义的，这里定义了一个截
%断频率为0.6的十阶低通滤波器和一个截断频率为0.5的十四阶低通滤波器
b1=remez(10,[0 0.4 0.6 1],[1 1 0 0]);
[H1,w]=freqz(b1,1,64,'whole');
colormap(jet(64))
plot(w/pi-1,fftshift(abs(H1)))
b2=remez(14,[0 0.3 0.5 1],[1 1 0 0]);
[H2,w]=freqz(b2,1,64,'whole');
colormap(jet(64))
plot(w/pi-1,fftshift(abs(H2)))
%用于频率变换的两个一维滤波器的频率响应如图7-18和图7-19所示
%然后，用默认的McClellan变换生成期望的近似圆对称的滤波器
h1=ftrans2(b1);
figure,freqz2(h1,[32 32])
```

```
h2=ftrans2(b2);
figure,freqz2(h2,[32 32])
```

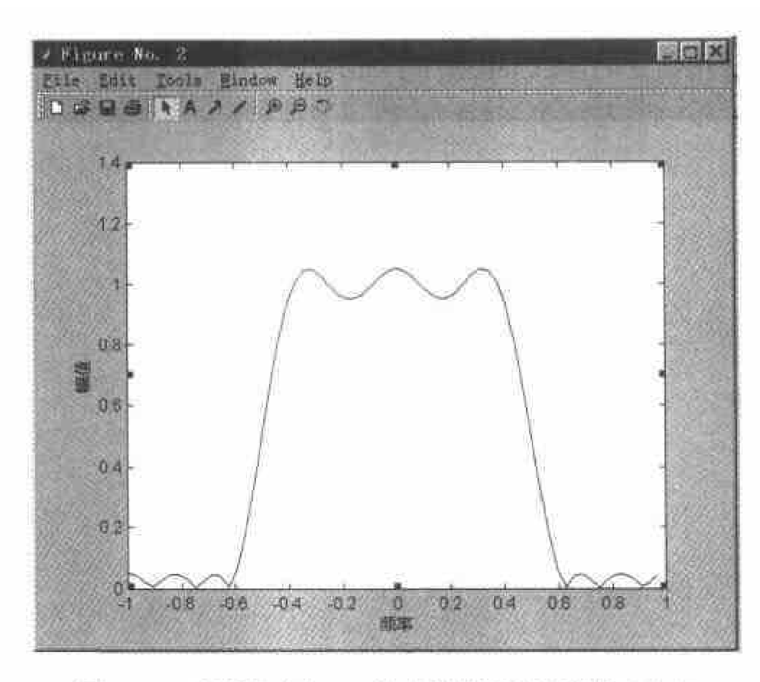


图 7-18 截断频率为 0.6 的十阶低通滤波器频率响应

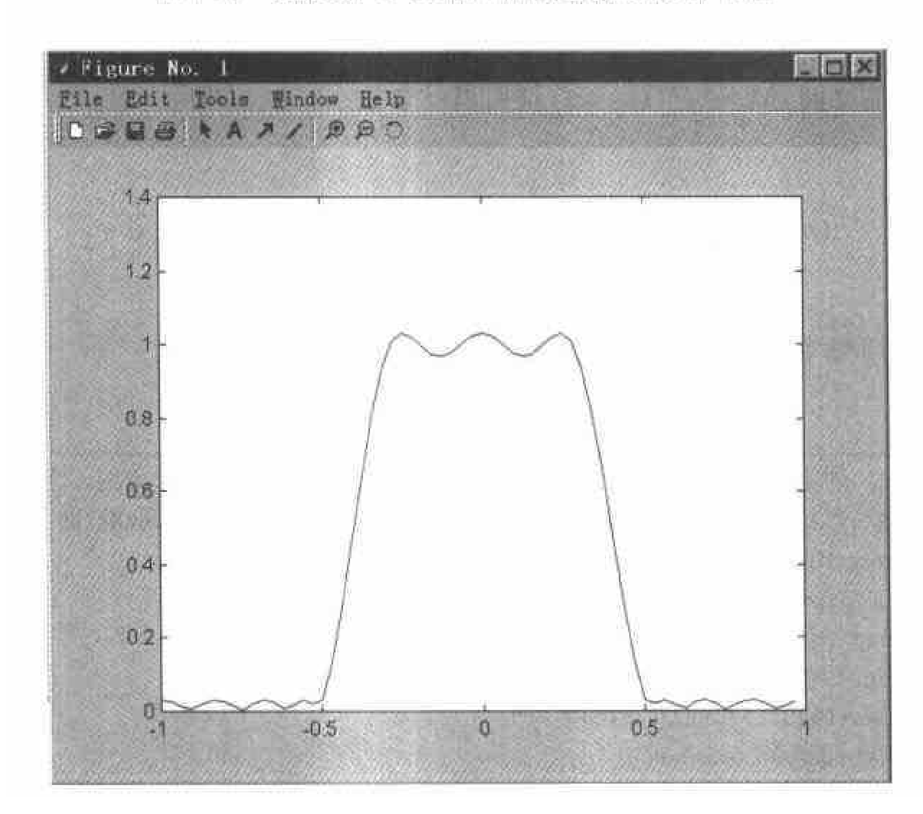


图 7-19 截断频率为 0.5 的十四阶低通滤波器

两个二维低通滤波器的频率响应如图 7-20 和图 7-21 所示。本例中为了显示差别只任意设计了两个截断频率和阶数都不同的低通滤波器，在实际应用中，阶数和截断频率都要根据实际情况而定，读者可以通过多次试验得到比较理想的滤波器。

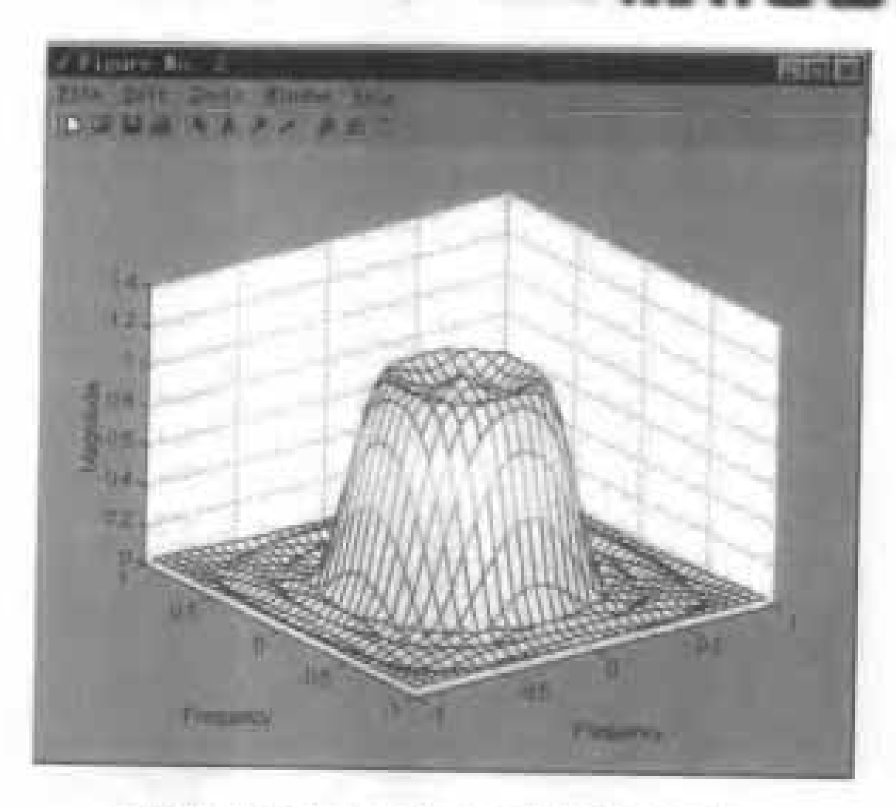


图 7-20　截断频率为 0.6 的十阶二维滤波器的频率响应

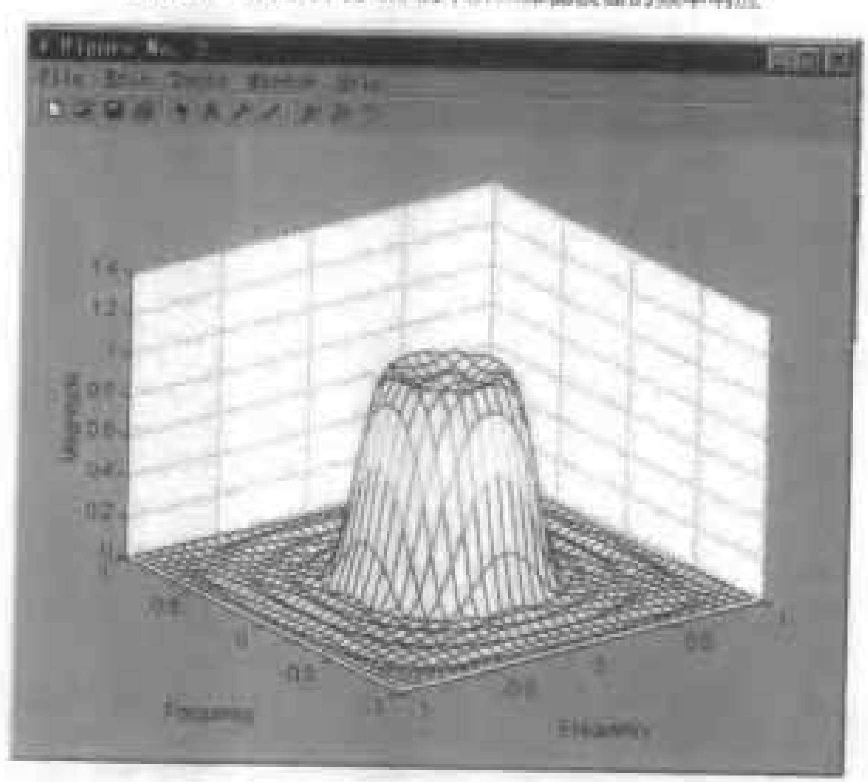


图 7-21　截断频率为 0.5 的十四阶二维滤波器的频率响应

第 8 章　图像增强原理及方法

图像增强是一类基本的图像处理技术，其目的是对图像进行加工，以得到对具体应用来说视觉效果更“好”、更“有用”的图像。这里的“好”和“有用”要因具体的应用目的和要求而异，并且所需要的具体增强技术也可不相同。

目前常用的增强技术根据其处理所进行的空间不同，可分为基于图像域的方法和基于变换域的方法。第一类，直接在图像所在的空间进行处理，也就是在像素组成的空间里直接对像素进行操作；第二类，在图像的变换域对图像进行间接处理。另外，在第一类方法里还可分为两组：一是基于像素（点）的，也就是对图像的每次处理是对每个像素进行的，增强过程对每个像素的处理与其他像素无关；二是基于模板的，也就是对图像的每次处理是对小的子图像（模板）进行的。

空域增强方法可表示为：

$$g(x, y) = EH[f(x, y)] \tag{8-1}$$

其中 $f(x,y)$ 和 $g(x,y)$ 分别为增强前后的图像，EH 代表增强操作。如果 EH 是定义在每个 (x,y) 上的，则 EH 是点操作；如果 EH 是定义在 (x,y) 的某个邻域上的，则 (x,y) 成为模板操作。如果模板最小时，可以是一个像素点，此时就变成了点操作，所以点操作可看做是模板操作的一个特例。

本章的各节中将分别介绍如何利用 MATLAB 实现以上各种图像处理技术。

本章主要内容：

- 空域变换增强
- 空域滤波增强
- 频域增强

8.1　空域变换增强

当 $g(x,y)$ 的值取决于在 (x,y) 处的 $f(x,y)$ 值，而 EH 就是一个灰度变换。如以 s 和 t 分别表示 $f(x,y)$ 和 $g(x,y)$ 在 (x,y) 位置处的灰度值，则此时式（8-1）可写成

$$t = EH(s) \tag{8-2}$$

基于点操作的方法也叫灰度变换，常见的方法有以下几类：

- 直接对每个像素进行操作。
- 借助直方图进行变换。
- 借助对一系列图像间的操作进行变换。

8.1.1 直接灰度调整

1. 增强对比度

增强对比度实际是增强原图的各部分的反差。实际中往往是通过增加原图中某两个灰度值之间的动态范围来实现的。典型的增强对比度的变换曲线如图 8-1 所示。

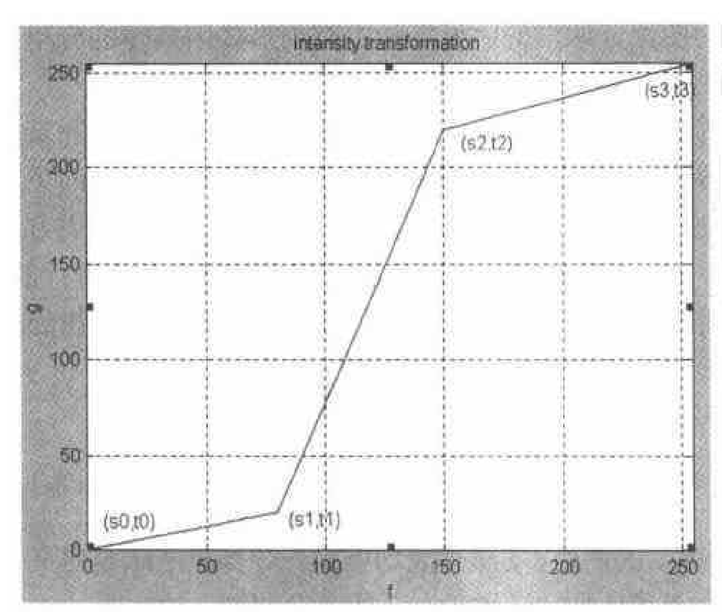


图 8-1 增强对比度的变换曲线



图 8-2 原始图像

从图中可以看出，通过这样一个变换，原图中灰度值在 0 到 s1 和 s2 到 255 之间的动态范围减小了，而原图中灰度值在 s1 到 s2 之间的动态范围增加了，从而这个范围内的对比度增加了。在实际应用中 s1、s2、t1、t2 可取不同的值进行组合，从而得到不同的效果。

例程 8-1 应用变换曲线增强对比度。

例程 8-1

```
X1=imread('pout.tif');
figure,imshow(X1)
f0=0;g0=0;
f1=70;g1=30;
f2=180;g2=230;
f3=255;g3=255;
%绘制变换曲线
figure,plot([f0,f1,f2,f3],[g0,g1,g2,g3])
axis tight,xlabel('f'),ylabel('g')
title('intensity transformation')

r1=(g1-g0)/(f1-f0);
b1=g0-r1*f0;
r2=(g2-g1)/(f2-f1);
```

```
b2=g1-r2*f1;
r3=(g3-g2)/(f3-f2);
b3=g2-r3*f2;
[m,n]=size(X1);
X2=double(X1);
%变换矩阵中的每个元素
for i=1:m
      for j=1:n
        f=X2(i,j);
        g(i,j)=0;
        if (f>=0)&(f<=f1)
        g(i,j)=r1*f+b1;
        elseif (f>=f1)&(f<=f2)
        g(i,j)=r2*f+b2;
        elseif (f>=f2)&(f<=f3)
        g(i,j)=r3*f+b3;
      end
   end
end
figure,imshow(mat2gray(g))
```

输出结果如图 8-3 和图 8-4 所示。



图 8-3　增强对比度所得的图像

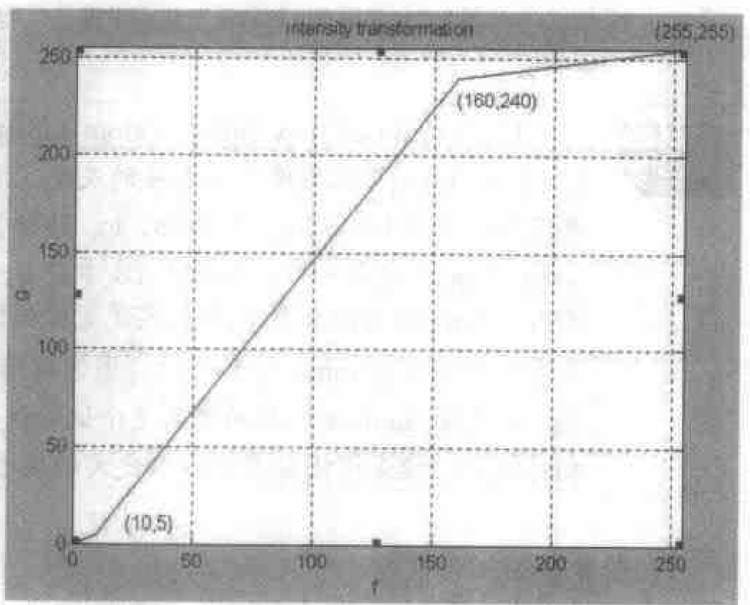


图 8-4　增强对比度的变换曲线

以上是在所选的参数模式下所得的运行结果。在处理过的图片上可以看出，小孩衣服上的图标和衣服之间的对比度增强了，更容易辨认了。通常在进行特定任务的模式识别工作之前，要对所要识别的图像进行增强对比度的处理，以使辨识工作更容易，辨识结果更准确。对于不同的图像，根据不同的情况，所选的参数可能有所不同，要根据实际情况而定。

例程 8-2 利用 MATLAB 工具箱中的函数 imadjust()增强对比度。

例程 8-2

```
X1=imread('pout.tif');
figure,imshow(X1)
J = imadjust(X1,[0.25 0.6],[],1.2322)
figure, imshow(J)
```

采用以上参数的变换曲线如图 8-5 所示。

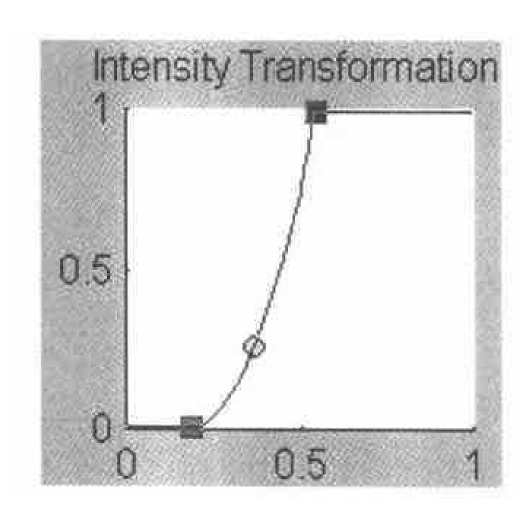


图 8-5 调用 MATLAB 工具箱中的函数增强对比度的变换曲线



J = imadjust(I,[low high],[bottom top],gamma)返回图像 I 经调整后的图像 J。[low high]为原图像中要变换的灰度范围，[bottom top]指定了变换后的灰度范围，其中 low、hig、bottom、top 的取值范围都是[0, 1]，并且 low<high，bottom<top，这是因为在 MATLAB 中将 0 ~ 255 的灰度范围映射到 0 ~ 1 的范围内。gamma 为矫正量 γ，它指定了变换曲线的形状，描述了 I 和 J 的值之间的关系；通常当 gamma 大于 1 时，图像变暗，而当 gamma 小于 1 时，图像变亮。上例中 gamma 所取的值只是个试验值，在这个取值下所得到的图像效果相对较好，读者根据需要可以取更大的值进行试验。

2. 图像求反

对图像求反是将原图的灰度值翻转，简单地说就是使黑变白，使白变黑，此时的变换曲线如图 8-6 所示，普通的黑白底片和照片就是这样的关系。具体的变换就是将图像中每个像素的灰度值根据变换曲线进行映射。

例程 8-3 应用变换曲线对图像求反。

例程 8-3

```
X1=imread('pout.tif');
figure,imshow(X1)
```

```
f1=200;%f1 和 g1 分别为 f、g 的最大值
g1=256;
figure,plot([0,f1],[g1,0])
axis tight,xlabel('f'),ylabel('g')
title('intensity transformation')

k=g1/f1;
[m,n]=size(X1);
X2=double(X1);
for i=1:m %转换矩阵的每个元素
      for j=1:n
          f=X2(i,j);
          g(i,j)=0;
          if (f>=0)&(f<=f1)
          g(i,j)=g1-k*f;
          else
          g(i,j)=0;
          end
      end
end
figure,imshow(mat2gray(g))
```

运行所得的结果如图 8-7 所示。

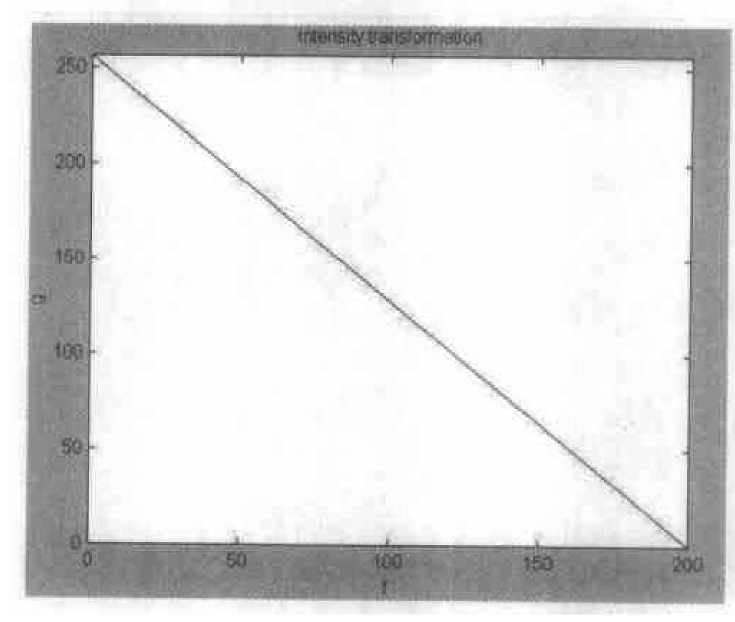


图 8-6　图像求反的变换曲线



图 8-7　图像求反所得的图像

3．动态范围压缩

这种方法的目标与增强对比度相反。当原图的动态范围太大，超出了某些显示设备所允许的动态范围时，如果直接使用原图则有一部分信息可能丢失，解决的办法是对原图进行灰度压缩。

采用对数形式的变换函数进行动态范围压缩：

$$g = c \times \log(1+f)$$

其中 c 是比例尺常数，见图 8-8 中所示的变换曲线。原始图像如图 8-9 所示。

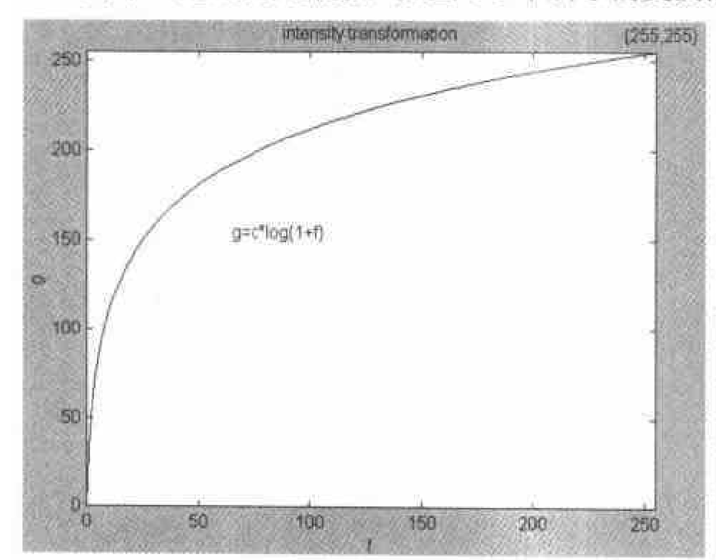


图 8-8　对数变换曲线

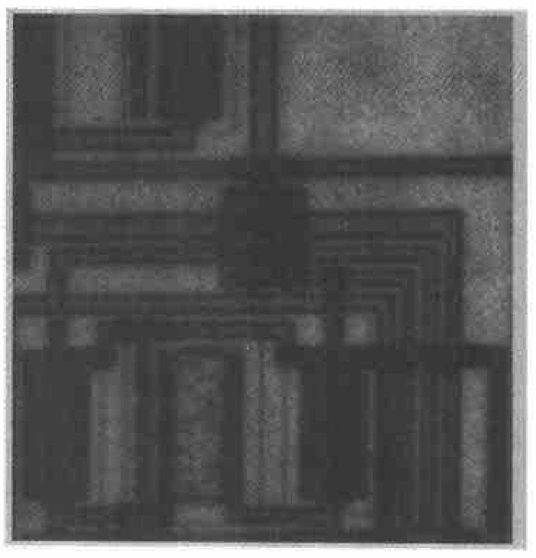


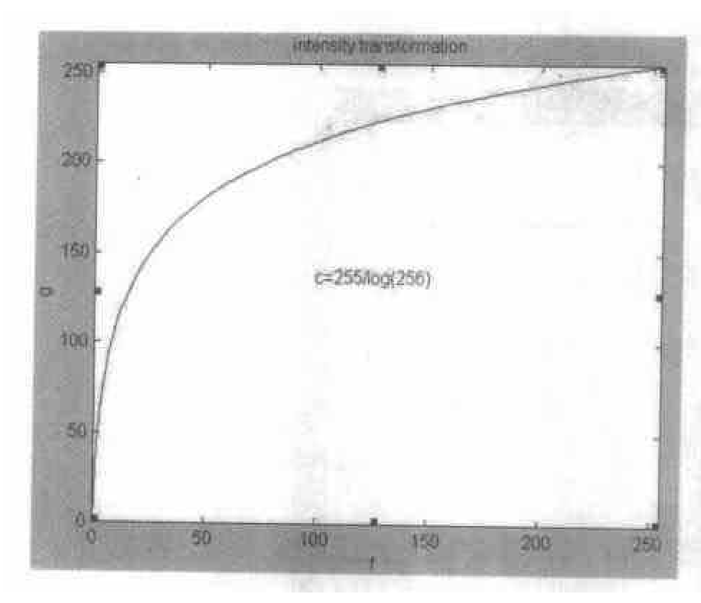
图 8-9　原始图像

例程 8-4 采用对数形式的变换函数进行动态范围压缩。

例程 8-4

```
X1=imread('circuit.tif');
figure,imshow(X1)
%绘制变换曲线，如图 8-10 所示
c=255/log(256);
x=0:1:255;
y=c*log(1+x);
figure,plot(x,y)
axis tight,xlabel('f'),ylabel('g')
title('intensity transformation')
%转换矩阵中的每个元素
[m,n]=size(X1);
X2=double(X1);
for i=1:m
    for j=1:n
        g(i,j)=c*log(X2(i,j)+1);
    end
end
figure,imshow(mat2gray(g))
```

当 c=255/log(256)时的运行结果如图 8-11 所示。根据所处理图像的不同情况，可选取不同的 c 值，当然处理所得的结果会有所不同，读者可从多次试验中选择符合应用标准的参数值。



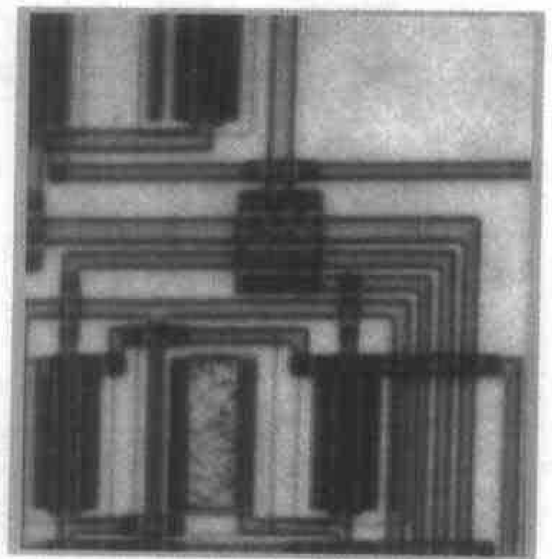


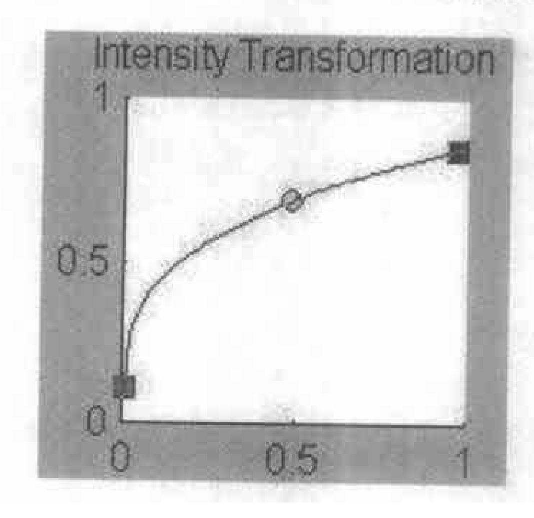
图 8-10　c=255/log(256)对数变换曲线　　　图 8-11　对数变换处理后的图像

例程 8-5 利用 MATLAB 工具箱中的函数 imadjust()进行变换。

例程 8-5

```
X1=imread('circuit.tifd');
figure,imshow(X1)
J = imadjust(X1,[],[ 0.1 0.8],0.35287)
figure, imshow(J)
```

采用以上参数的变换曲线和处理结果分别如图 8-12 和图 8-13 所示。





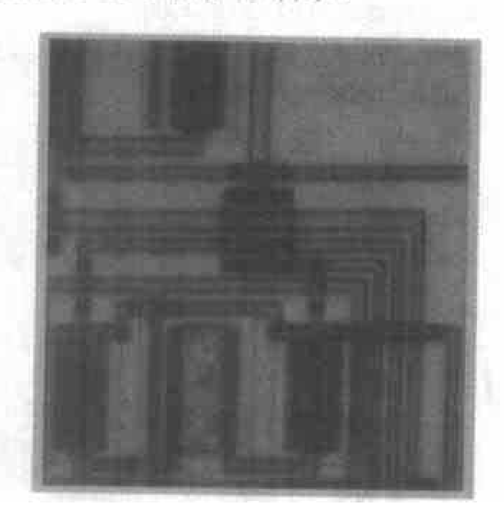


图 8-12 变换曲线　　　图 8-13 变换的图像

读者同样可以使用不同的参数组合来得到不同的效果。

4. **灰度切分**

灰度切分的目的与增强对比度相似，可以将某个灰度值范围变得比较突出。所不同的是，这里将所要突出的灰度范围变换成较高的灰度值，而将其余灰度值变换为较低的灰度

值。常用的变换曲线如图 8-14 所示。

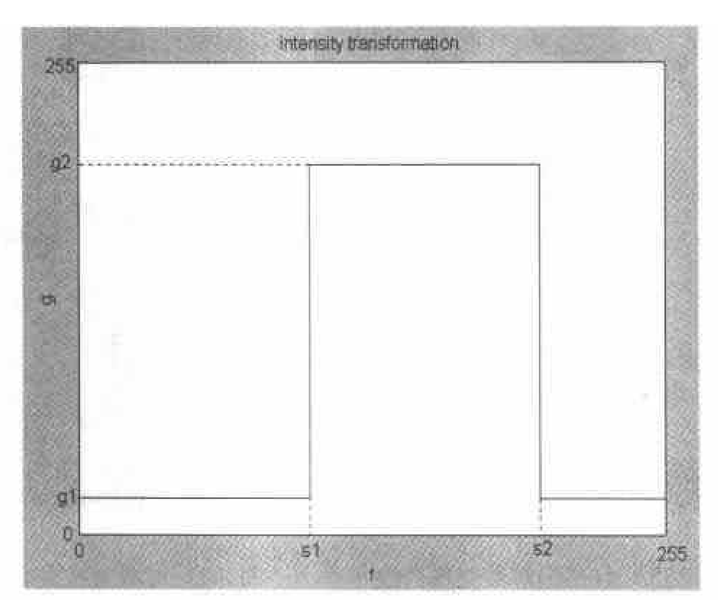


图 8-14　灰度切分的曲线表示图

其中的 s1、s2 为用户选定的参数，而 s1 和 s2 之间的灰度范围是用户想要突出的部分。例程 8-6 取 s1=100，s2=200。

例程 8-6

```
X1=imread('pout.tif');
figure,imshow(X1)
%绘制变换曲线
s1=100;s2=200;g1=20;g2=200;
figure,plot([0,s1,s1,s2,s2,255],[g1,g1,g2,g2,g1,g1])
axis tight,xlabel('f'),ylabel('g')
title('intensity transformation')
[m,n]=size(X1);
X2=double(X1);
for i=1:m    %转换矩阵中的每个元素
    for j=1:n
        f=X2(i,j);
        g(i,j)=0;
        if (f>=s1)&(f<=s2)
            g(i,j)=g2;
        else
            g(i,j)=g1;
        end
    end
end
figure,imshow(mat2gray(g))
```

变换曲线和运行结果分别如图 8-15 和图 8-16 所示。

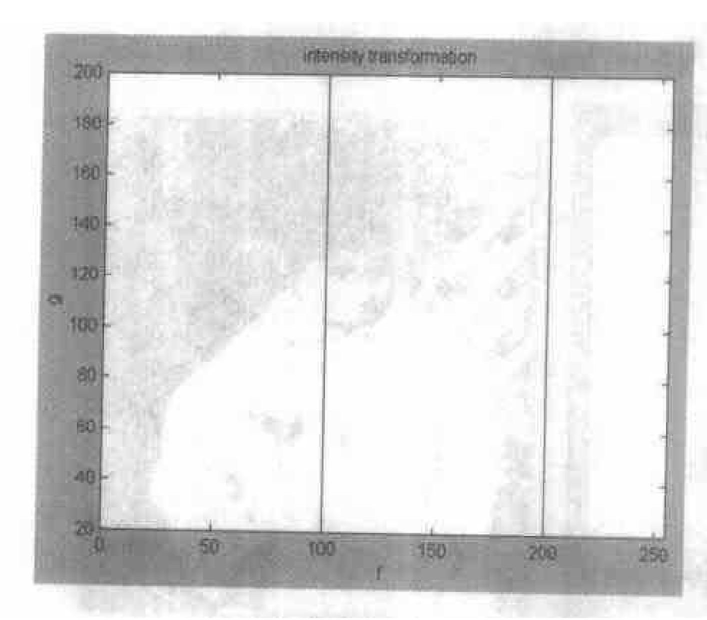


图 8-15 s1=100，s2=200 时灰度切分的变换曲线



图 8-16 灰度切分的结果

图像二值化可以看做是灰度切分的一种特殊情况，它将灰度值小于某个 f1 的所有像素点都置为 0，而将灰度值大于 f1 的所有像素点都置为 255；如果再将 0 和 255 这两个灰度级分别映射为 0 和 1，就得到了二值图像。

例程 8-7 取 f1=100。

例程 8-7

```
X1=imread('pout.tif');
figure,imshow(X1)
%绘制变换曲线
f1=100;
figure,plot([0,f1,f1,255],[0,0,255,255])
axis tight,xlabel('f'),ylabel('g')
title('intensity transformation')
%
[m,n]=size(X1);
X2=double(X1);
for i=1:m
    for j=1:n
        f=X2(i,j);
        g(i,j)=0;
        if (f>=0)&(f<=f1)
          g(i,j)=0;
        elseif (f>=f1)&(f<=255)
          g(i,j)=255;
        end
    end
end
figure,imshow(mat2gray(g))
```

变换曲线和变换结果分别如图 8-17 和图 8-18 所示。

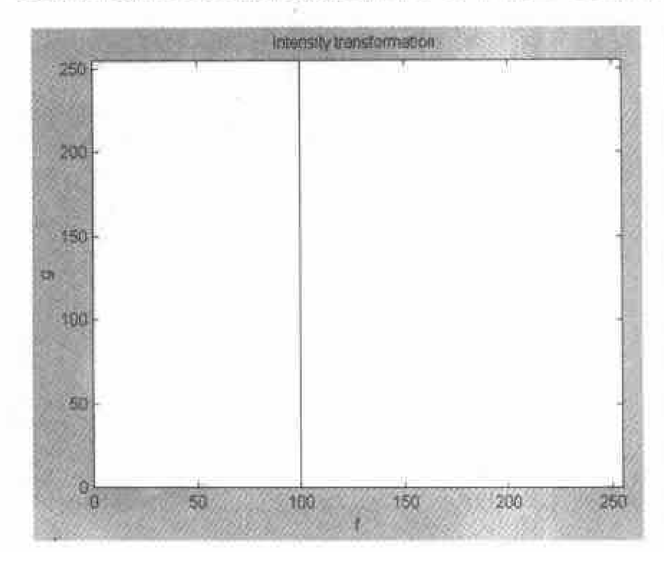




图 8-17　f1=100 时的二值变换曲线



图 8-18　二值化的结果

8.1.2　直方图处理

图像的灰度统计直方图是一个一维离散函数：

$$p(s_k) = n_k / n \qquad k=0，1，\cdots，L-1 \qquad (8-3)$$

上式中 S_k 是图像 f（x,y）的第 k 级灰度值，n_k 是 f（x,y）中具有灰度值 S_k 的像素的个数，n 是图像中像素的总数。由定义式可知，P（S_k）给出了对 S_k 出现概率的一个估计，所以直方图表明了图像中灰度值的分布情况。因此可以通过改变直方图的形状来达到增强图像对比度的效果。这种方法是以概率论为基础的。常用的方法有直方图均衡化和直方图规定化两种。图 8-19 至图 8-26 是 8 幅不同效果的图像及对应的直方图。通过对比可以看出，偏暗的图像灰度范围很窄，而且主要集中在低灰度级上；偏亮的图像灰度范围主要集中在高灰度级上；灰蒙蒙的图像同样是因为灰度范围窄，动态范围小，主要集中在中间的灰度级上；正常的图像动态范围很大，在各个灰度级上都有像素，从而图像看起来对比度较大，细节清晰。后面将要提到的直方图均衡化和直方图规定化主要就是基于此种分析对图像的直方图进行操作，来达到图像增强的目的。

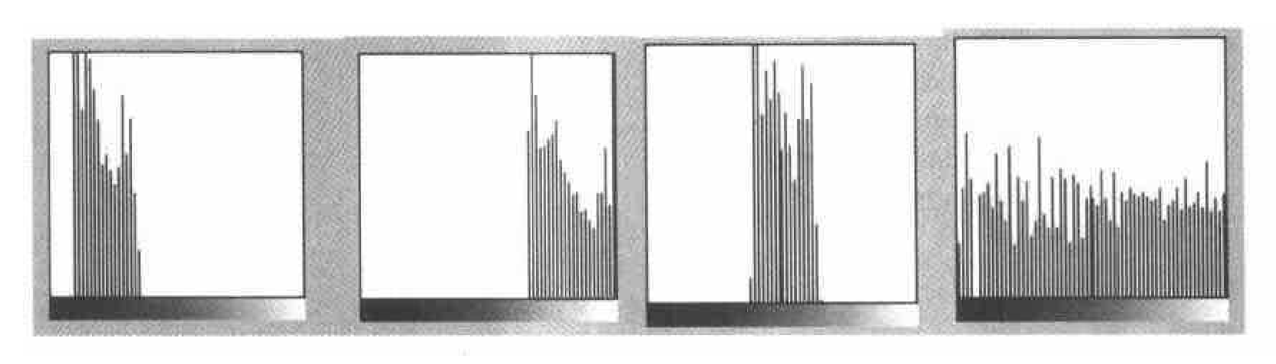


图 8-19　偏暗的　　图 8-20　偏亮的　　图 8-21　动态范围偏小　　图 8-22　正常的

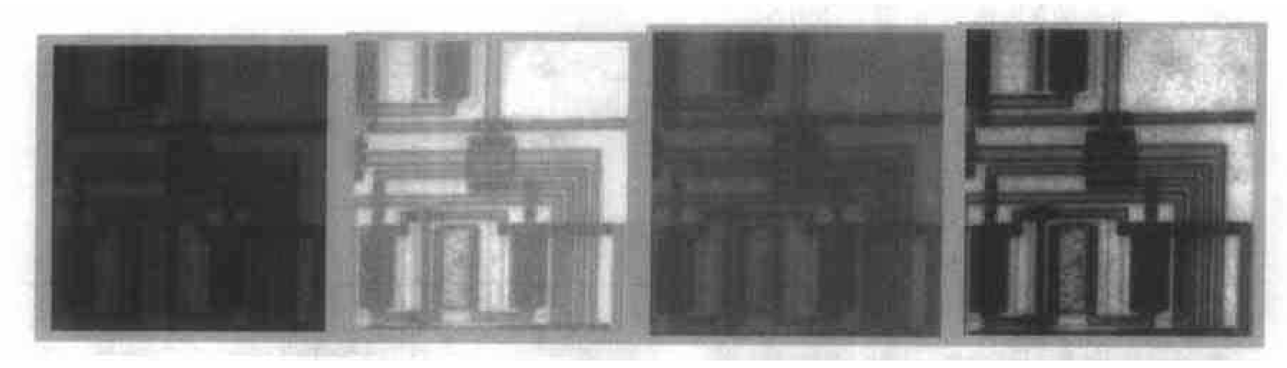


图 8-23　偏暗的　　图 8-24　偏亮的　　图 8-25　动态范围偏小　　图 8-26　正常的

1．**直方图均衡化**

这个方法的基本思想是把原始图像的直方图变换成均匀分布的形式，这样就增加了像素灰度值的动态范围，从而达到了增强图像整体对比度的效果。具体方法是：

（1）给出原始图像的所有灰度级 S_k，k=0,1,⋯,L-1。

（2）统计原始图像各灰度级的像素数 n_k。

（3）用式（8-3）计算原始图像的直方图。

（4）计算原始图像的累计直方图：

$$t_k = EH(s_k) = \sum_{i=0}^{k} \frac{n_i}{n} = \sum_{i=0}^{k} p_s(s_i) \qquad \begin{matrix} 0 \leqslant S_k \leqslant 1 \\ k = 0,1,\cdots,\mathrm{L}-1 \end{matrix} \qquad (8\text{-}4)$$

（5）取整计算：

$$t_k = int[(\mathrm{N}-1)t_k + k/\mathrm{N}]$$

（6）确定映射关系 $s_k \rightarrow t_k$。

（7）统计新直方图各灰度级的像素数 n_k。

（8）计算新的直方图：

$$p_t(t_k) = n_k / n$$

例程 8-8 实现直方图均衡化。

例程 8-8 直方图均衡化

```
%这里有一幅图像，如图 8-27 所示，其动态范围较小，而且较暗，反映在直方图上，
%如图 8-28 所示，就是其直方图所占据的灰度值范围比较窄，而且集中在低灰度值一边。
%为了使图像更清晰，我们采用直方图均衡化的方法来增加图像灰度动态范围，增强对比度。
%在 MATLAB 中，可以直接调用 J=histeq(I,n)函数来完成这项工作，
%其中 I 是原始图像矩阵，J 是变换后所得的图像矩阵。用户可以指定均衡化
```

```
%后的灰度级数 n，默认值为 64。
I = imread('tire.tif');
J = histeq(I);
imshow(I)
figure, imshow(J)
figure,imhist(I,64)
figure,imhist(J,64)
```



图 8-27　原始图像

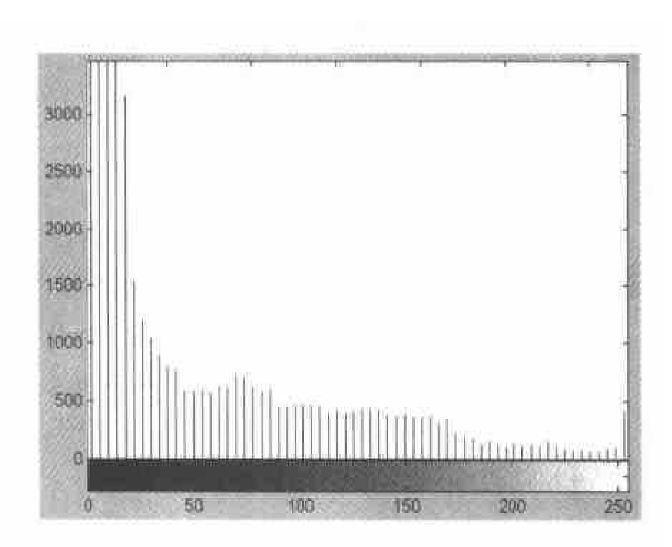


图 8-28　原始直方图

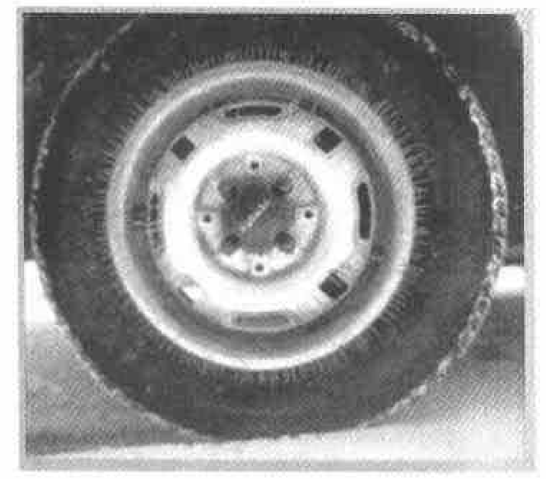


图 8-29　直方图均衡化所得的图像

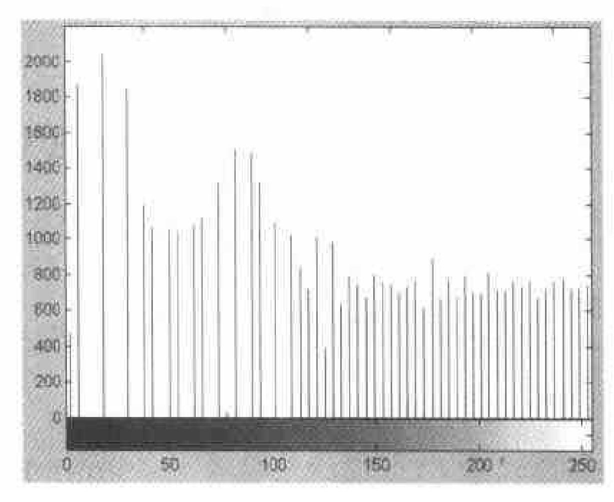


图 8-30　直方图均衡化后的直方图

原始图像、原始直方图、处理后的图像和直方图分别如图 8-27、图 8-28、图 8-29、图 8-30 所示。从处理前后的图像可以看出，许多在原始图像中看不清的细节在直方图均衡化处理后所得的图像中都变得十分清晰。

2．直方图规定化

直方图均衡化的优点是能自动增强整个图像的对比度，但它的具体增强效果不易控制，处理的结果总是得到全局均衡化的直方图。实际中有时需要变换直方图使之成为某个特定的形状，从而有选择地增强某个灰度值范围的对比度。这时可采用比较灵活的直方图规定化。直方图规定化方法主要有三个步骤：

（1）对原始图像的直方图进行灰度均衡化：

$$t_k = EH_s(s_i) = \sum_{i=0}^{k} p_s(s_i) \qquad k = 0,1,\cdots,M-1 \qquad (8\text{-}5)$$

（2）规定所需要的直方图，并计算能使规定的直方图均衡化的变换：

$$v_l = EH_u(u_j) = \sum_{j=0}^{l} p_u(u_j) \qquad j = 0,1,\cdots,N-1 \qquad (8\text{-}6)$$

（3）将第一步所得到的变换翻转过来，也就是将原始图像对应映射到规定的直方图，将所有的 $p_s(s_i)$ 对应到 $p_u(u_j)$ 去。对应规则可选用组映射规则（group mapping law，GML）。设有一个整数函数 $I(l)$，$l=0,1,\cdots,N-1$ 满足 $0 \leqslant I(0) \leqslant \cdots \leqslant I(l) \leqslant \cdots \leqslant I(N-1) \leqslant M-1$。现在要确定能使下式达到最小的 $I(l)$：

$$\left| \sum_{i=0}^{I(l)} p_s(s_i) - \sum_{j=0}^{l} p_j(u_j) \right| \qquad l = 0,1,\cdots,N-1 \qquad (8\text{-}7)$$

如果 $l = 0$，则将其 i 从 0 到 I（0）的 $p_s(s_i)$ 对应到 $p_u(u_0)$ 去；如果 $l \geqslant 1$，则将其 i 从 $I(l-1)+1$ 到 $I(l)$ 的 $p_s(s_i)$ 对应到 $p_u(u_j)$ 去。



在 MATLAB 中调用函数 J = histeq(I,hgram)可以实现直方图规定化。其中 hgram 是由用户指定的矢量，规定将原始图像 I 的直方图近似变换成 hgram，hgram 中的每一个元素都在[0,1]中。

例程 8-9 实现直方图规定化。

例程 8-9

```
%采用图 8-31 所示的规定化函数
I = imread('tire.tif');
hgram=0:255
J = histeq(I,hgram);
imshow(I)
figure, imshow(J)
figure,imhist(I,64)
figure,imhist(J,64)
```

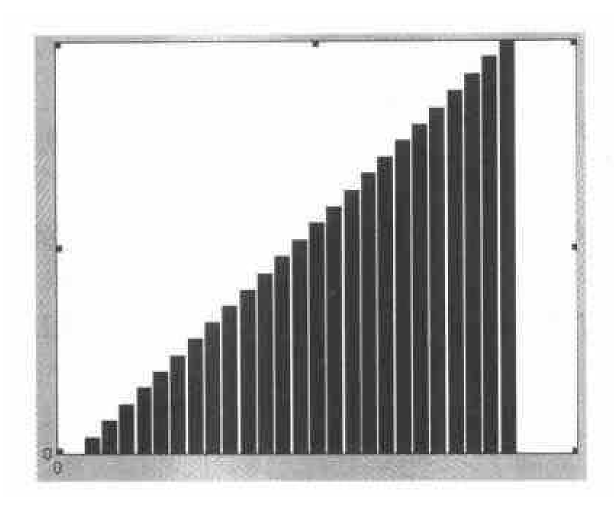


图 8-31　规定化函数

直方图规定化所得的图像和直方图如图 8-32 和图 8-33 所示。比较图 8-29 和图 8-32 可知，使用直方图均衡化得到的结果在一些较暗的区域有些细节仍不太清楚。采用图 8-31 所示的规定化函数对同一幅图像进行直方图规定化变换，所得的结果比均衡化更亮，对应于均衡化图像中较暗区域的一些细节更清晰。从直方图上看，高灰度值一侧更为密集。



图 8-32　直方图规定化所得的图像

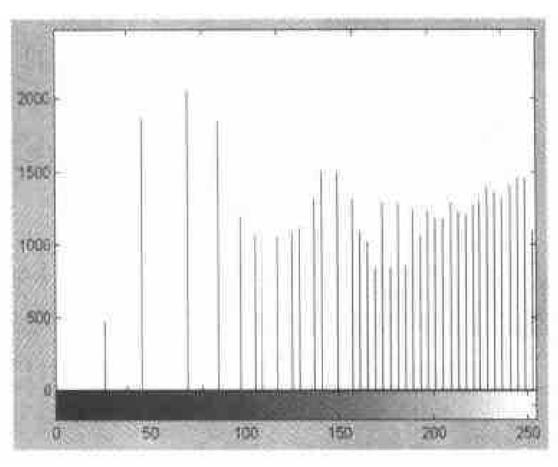


图 8-33　直方图规定化后的直方图

8.1.3　图像间的代数运算

有些图像增强技术是靠对多幅图像进行图像间的代数运算实现的。代数运算是指对两幅图像进行点对点的加、减、乘、除计算而得到输出图像的运算。对于相加和相乘的情形，可能不只有两幅图像参加运算。常用的图像间的代数运算有图像相减和图像相加。

1．图像相减运算

设有图像 $f(x, y)$ 和 $h(x, y)$，它们的差为：

$$g(x, y) = f(x, y) - h(x, y)$$



图像相减可以用于去除一幅图像中所不需要的加性图案，加性图案可能是缓慢变化的背景阴影、周期性的噪声，或在图像上每个像素处均已知的附加污染等。减法也可以用于检测同一场景的两幅图像之间的变化。例如通过对一场景的序列图像的减运算可检测运动。在计算用于确定物体边界位置的梯度时，也要用到图像减运算。

例程 8-10 显示采用离散余弦变换压缩的图像与原图像的差别。

例程 8-10

```
%用原图像与压缩后的图像相减
I = imread('cameraman.tif');
I = double(I)/255;
%计算离散变换矩阵，返回结果为双精度型
T = dctmtx(8);
%实现图像的显示块操作
B = blkproc(I,[8 8],'P1*x*P2',T,T');
mask =[1 1 1 1 0 0 0 0
       1 1 1 0 0 0 0 0
       1 1 0 0 0 0 0 0
       1 0 0 0 0 0 0 0
       0 0 0 0 0 0 0 0
       0 0 0 0 0 0 0 0
       0 0 0 0 0 0 0 0
       0 0 0 0 0 0 0 0];
B2 = blkproc(B,[8 8],'P1.*x',mask);
I2 = blkproc(B2,[8 8],'P1*x*P2',T',T);
imshow(I), figure, imshow(I2)
M=I2-I;
figure, imshow(mat2gray(M))
```

压缩前、后的图像及相减的图像分别如图 8-34、图 8-35 和图 8-36 所示。



图 8-34　原始图像



图 8-35　经压缩、解压后的图像

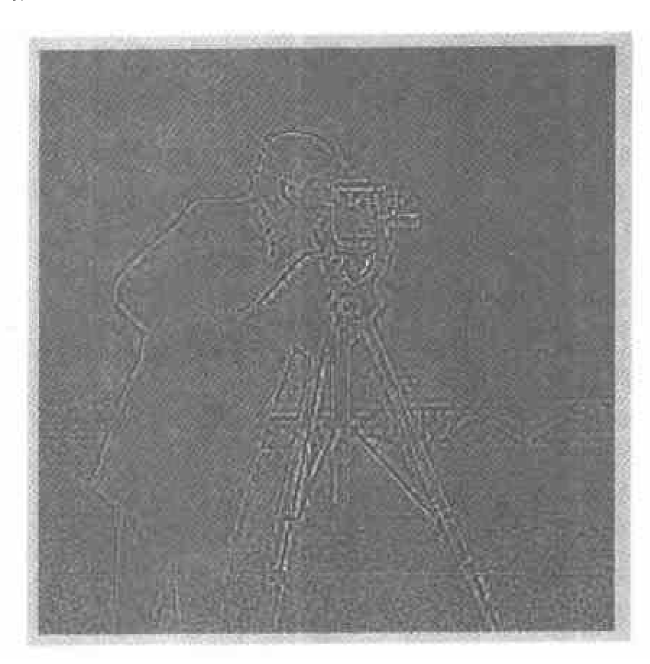


图 8-36　相减所得的图像

2．图像相加运算

设有图像 $f(x,y)$ 和 $h(x,y)$，它们的和为：

$$g(x,y)=f(x,y)+h(x,y)$$

图像相加的一个重要应用是对同一场景的多幅图像求平均值。这种方法经常用来有效地降低加性随机噪声的影响。图像相加也可以用来将一幅图像的内容叠加到另一幅图像上去，以达到二次曝光的效果。

下面以 saturn.tif 为例，将多幅加入高斯噪声的图像经过相加，求平均，达到去除噪声的目的。

例程 8-11 实现图像相加。

例程 8-11

```
I=imread(saturn.tif);
figure,imshow(I)
[m,n]=size(I);
J(m,n)=0;
for i=1:N
X=imnoise(I,'gaussian');
    figure,imshow(X)
    Y=double(X);
    J=J+Y/N;
end
figure,imshow(mat2gray(J))
```

程序中 N 是设定噪声图像的个数。原始图像、3 幅随机加入高斯噪声的不同的图像、10 幅噪声图平均得到的结果、100 幅噪声图平均得到的结果分别如图 8-37 至图 8-42 所示。



图 8-37　原始 saturn.tif 图像

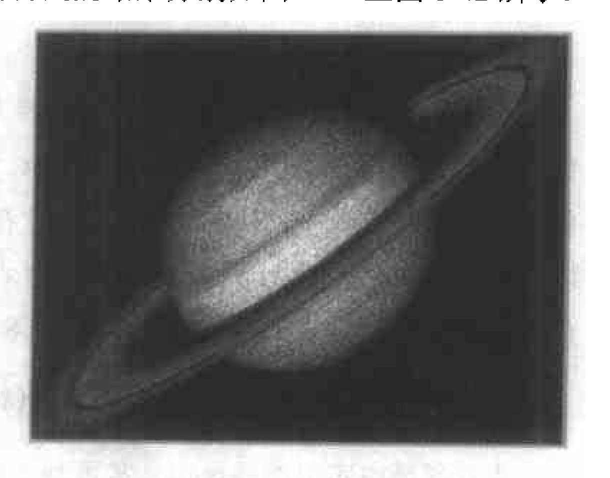


图 8-38　加噪图像 1

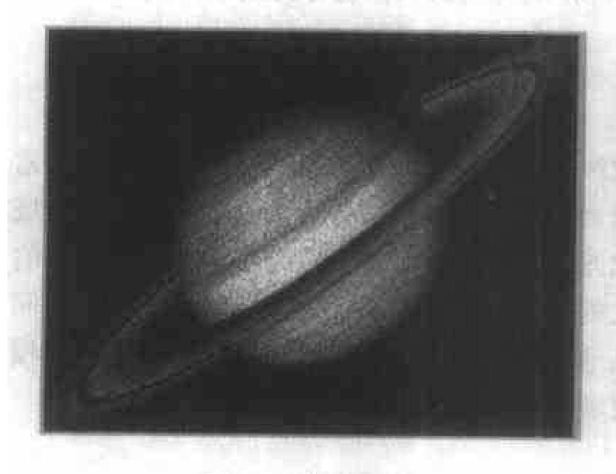


图 8-39　加噪图像 2

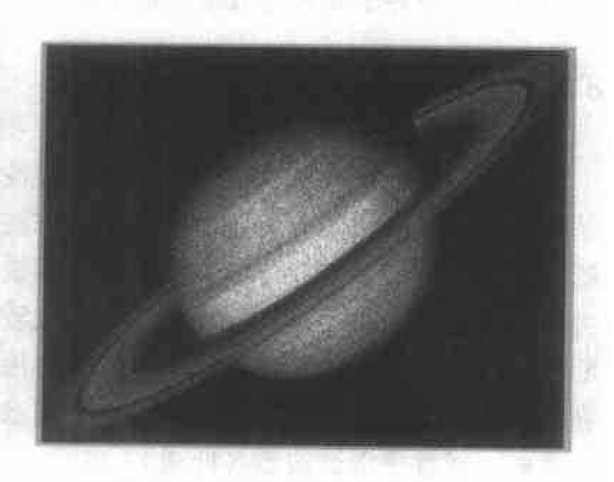


图 8-40　加噪图像 3

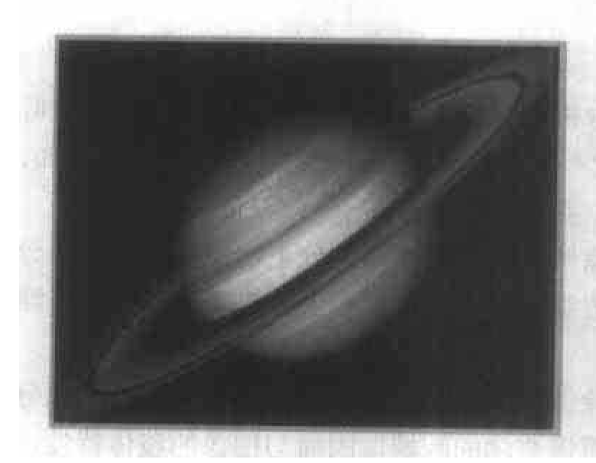


图 8-41　100 幅噪声图平均得到的结果

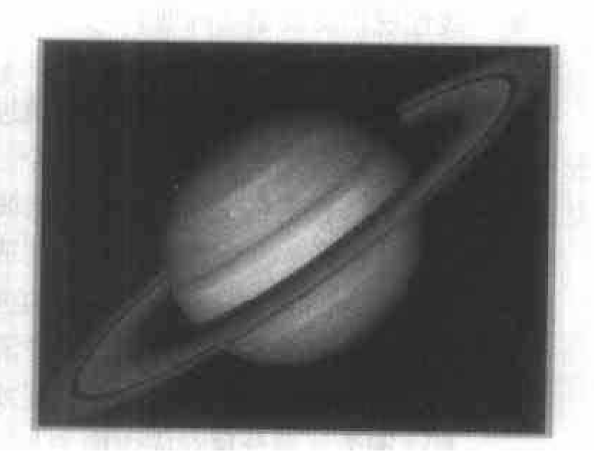


图 8-42　10 幅噪声图平均得到的结果

将各个结果比较可知，参加平均的噪声图像越多，去除噪声的效果越好。

8.2 空域滤波增强

一般情况下，像素的邻域比该像素要大，也就是说这个像素的邻域中除了本身以外还包括了其他像素。在这种情况下，$g(x,y)$ 在 (x,y) 位置处的值不仅取决于 $f(x,y)$ 在 (x,y) 位置处的值，而且取决于 $f(x,y)$ 在以 (x,y) 为中心的邻域内所有像素的值。如仍以 s 和 t 分别表示 $f(x,y)$ 和 $g(x,y)$ 在 (x,y) 位置处的灰度值，并以 $n(s)$ 代表 $f(x,y)$ 在 (x,y) 邻域内像素的灰度值，则此时式（8-1）可写成：

$$t = EH[s, n(s)] \tag{8-8}$$

为在邻域内实现增强操作，常可利用模板与图像进行卷积。每个模板实际上是一个二维数组，其中各个元素的取值确定了模板的功能，这种模板操作也称为空域滤波。

8.2.1 基本原理

根据其特点，空域滤波一般可分为线性滤波和非线性滤波两类。线性滤波器的设计常基于对傅里叶变换的分析。非线性空域滤波器则一般直接对邻域进行操作。另外各种空域滤波器根据功能又主要分成平滑滤波器和锐化滤波器。平滑可用低通来实现。平滑的目的可分为两类：一类是模糊，目的是在提取较大的目标前去除太小的细节或将目标内的小间断连接起来；另一类是消除噪声。锐化可用高通滤波来实现。锐化的目的是为了增强被模糊的细节。结合这两种分类法，可将空间滤波增强方法分成 4 类：

- 线性平滑滤波器（低通）
- 非线性平滑滤波器（低通）
- 线性锐化滤波器（高通）
- 非线性锐化滤波器（高通）

空域滤波器的工作原理都可借助频域进行分析。它们的基本特点都是让图像在傅里叶空间的某个范围的分量受到抑制，而让其他分量不受影响，从而改变输出图像的频率分布，达到增强的目的。在增强中用到的空间滤波器主要有两类：

- 平滑（低通）滤波器：它能减弱或消除傅里叶空间的高频分量，但不影响低频分量。因为高频分量对应图像中的区域边缘等灰度值具有较大较快变化的部分，滤波器将这些分量滤去可使图像平滑。
- 锐化（高通）滤波器：它能减弱或消除傅里叶空间的低频分量，但不影响高频分量。因为低频分量对应图像中灰度值缓慢变化的区域，因而与图像的整体特性，如整体对比度和平均灰度值等有关，将这些分量滤去可使图像锐化。

空域滤波器都是利用模板卷积，主要步骤是：

（1）将模板在图中漫游，并将模板中心与图中某个像素位置重合。

（2）将模板上的系数与模板下对应的像素相乘。



（3）将所有乘积相加。

（4）将和（模板的输出响应）赋给图中对应模板中心位置的像素。

图 8-43（a）中给出了一幅图像的一部分，其中所标的是一些像素的灰度值。现设有一个 3×3 的模板如图 8-43（b）所示，模板中所标的是模板系数。如将 k_0 所在位置与图灰度值为 s_0 的像素重合（也就是将模板中心放在图中 (x, y) 位置），模板的输出响应 R 为：

$$R = k_0 s_0 + k_1 s_1 + \cdots + k_8 s_8 \tag{8-9}$$

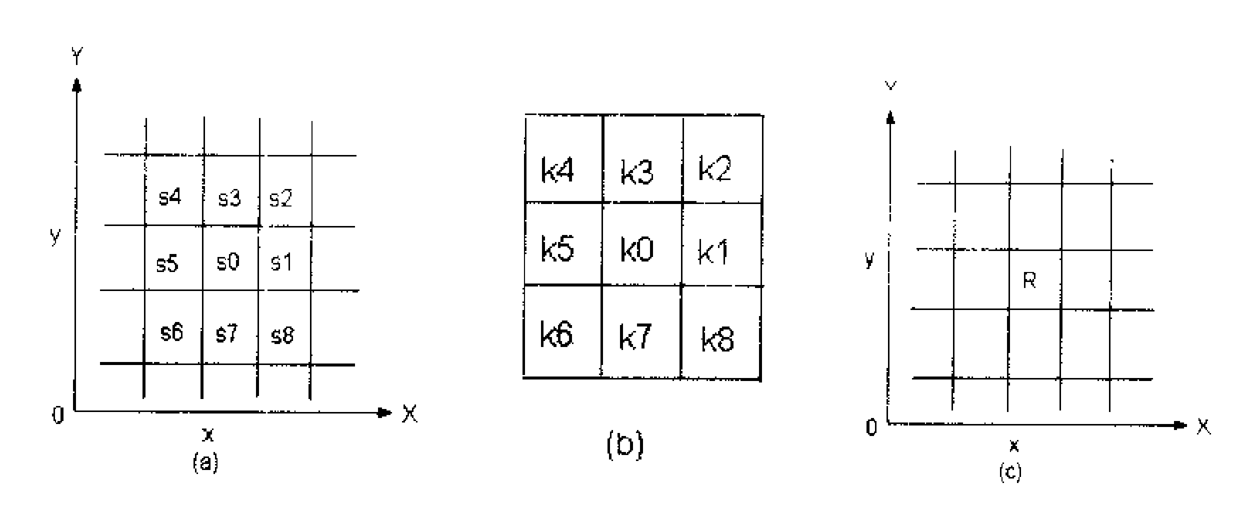


图 8-43　用 3×3 的模板进行空域滤波的示意图

将 R 赋给增强图，作为在 (x, y) 位置的灰度值，如图 8-43（c）所示。如果对原图中每个像素都进行这样的操作，就可得到增强图所有位置的新灰度值。如果我们在设计滤波器时给各个 k 赋不同的值，就可得到不同的高通或低通效果。下面分别介绍在 MATLAB 中如何应用平滑和锐化滤波器。

8.2.2　平滑滤波器

1. 线性平滑滤波器

线性低通滤波器是最常用的线性平滑滤波器。这种滤波器的所有系数都是正的。对 3×3 的模板来说，最简单的操作是取所有系数都为 1。为保证输出图像仍在原来的灰度范围内，在计算出 R 后要将其除以 9 再进行赋值。这种方法称为邻域平均法。

例程 8-12 实现均值过滤器。

例程 8-12

```
%以 saturn.tif 为例，加入椒盐噪声，并在 MATLAB 中调用 B = filter2(h,A)实现均值过滤器
I=imread('eight.tif');
```

```
J=imnoise(I,'salt &pepper',0.02);
imshow(I)
figure,imshow(J)
K1=filter2(fspecial('average',3),J)/255;
K2=filter2(fspecial('average',5),J)/255;
K3=filter2(fspecial('average',7),J)/255;
figure,imshow(K1)
figure,imshow(K2)
figure,imshow(K3)
```



其中 B = filter2(h,A)返回图像 A 经算子 h 滤波的结果。fspecial('average',3)用来创建 3×3 的均值滤波器，fspecial('average',5) 用来创建 5×5 的均值滤波器，fspecial('average',7) 用来创建 7×7 的均值滤波器，其中 3×3 为默认模板。

加入椒盐噪声的图像和均值滤波的图像分别如图 8-44、图 8-45、图 8-46 和图 8-47 所示。

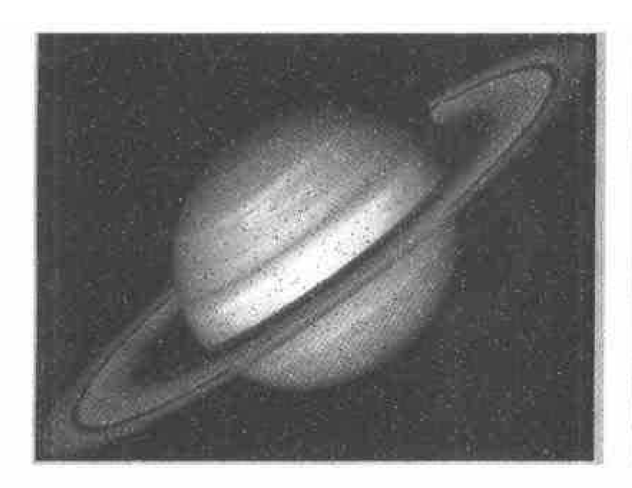


图 8-44 加入椒盐噪声的图像

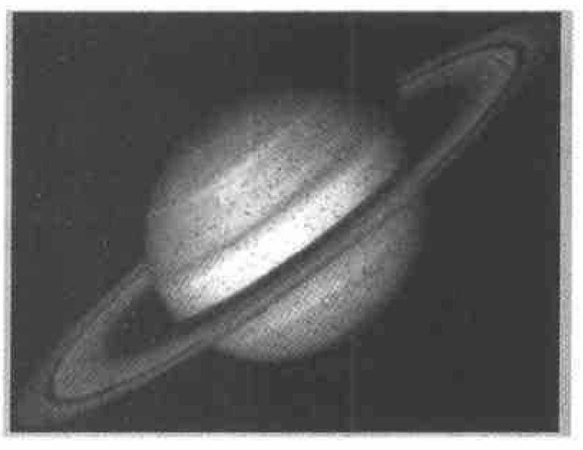


图 8-45 3×3 的均值滤波器处理结果

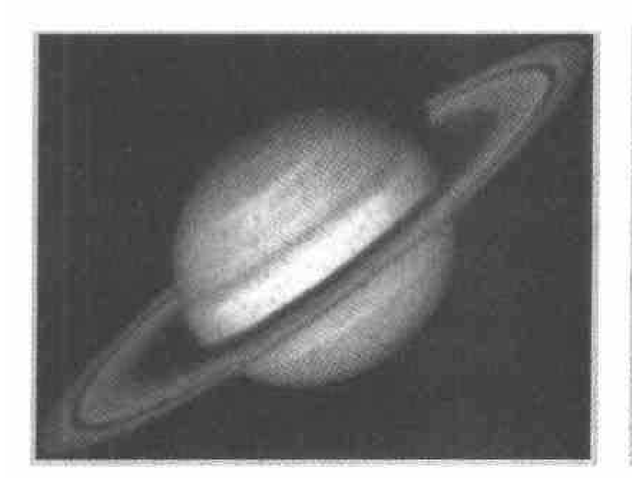


图 8-46 5×5 的均值滤波器处理结果

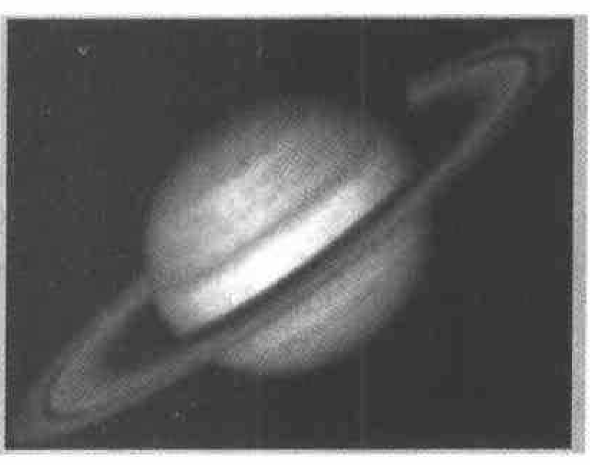


图 8-47 7×7 的均值滤波器处理结果



比较以上采用不同尺寸的均值滤波器进行低通滤波处理的结果可知，当所用的平滑模板的尺寸增大时，消除噪声的效果增强，但同时所得的图像变得更模糊，细节的锐化程度逐步减弱。

此外，维纳滤波器也是经典的线性降噪滤波器。维纳滤波是一种自适应滤波，它能根据图像的局部方差调整滤波器的输出。局部方差越大，滤波器的平滑作用越强。它的最终目标是使恢复图像 $f^*(x,y)$ 与原始图像 $f(x,y)$ 的均方误差

$$e^2 = E[(f(x,y) - f^*(x,y))^2]$$

最小。

例程 8-13 实现维纳滤波器。

例程 8-13

```
%在 MATLAB 中调用 J = wiener2(I,[m n],noise)函数
I = imread('saturn.tif');
J = imnoise(I,' gaussian', 0,0.005);
K1 = wiener2(J, [5,5]);
figure,imshow(K1)
```



在 J = wiener2(I,[m n],noise)函数中，[m n]指定了滤波器的窗口大小为 m×n，默认值为 3×3。noise 指定了噪声的功率，默认的是加性噪声（高斯白噪声）。

Wiener2 函数非常适用于图像中的白噪声（比如高斯噪声），加入高斯噪声的图像和维纳滤波的效果分别如图 8-48 和图 8-49 所示。

高斯滤波器在空域中也是具有平滑性能的低通滤波器，可通过调用 h = fspecial('gaussian')函数实现高斯滤波器。

例程 8-14 实现高斯滤波器。

例程 8-14

```
I = imread('saturn.tif');
J = imnoise(I,'gaussian', 0,0.005);
figure,imshow(J)
h = fspecial( gaussian');
K=filter2(h,J)/255;
figure,imshow(K)
```

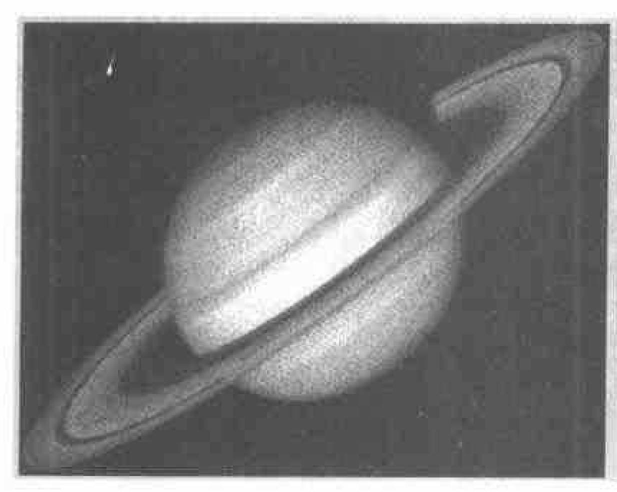


图 8-48　加入高斯噪声的图像

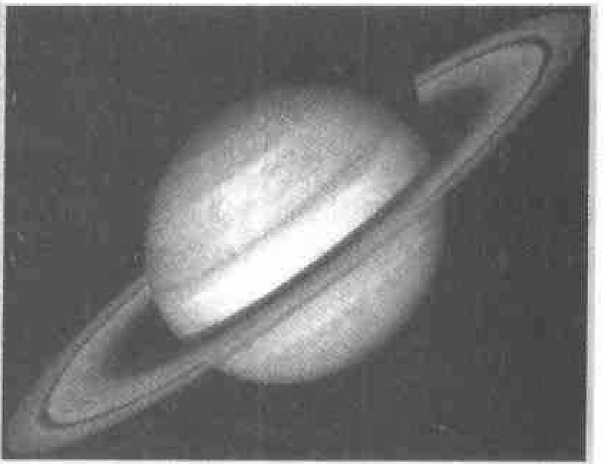


图 8-49　维纳滤波的结果

加入高斯噪声的图像和高斯滤波的结果如图 8-50 和图 8-51 所示。

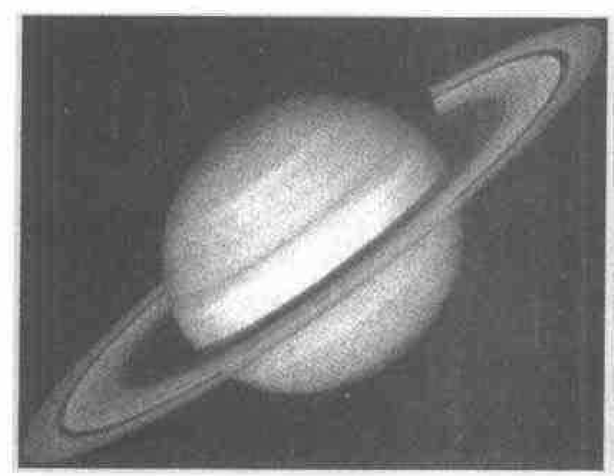


图 8-50　加入高斯噪声的图像

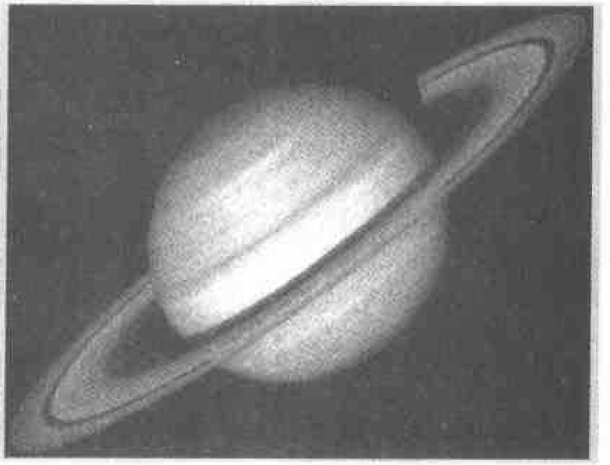


图 8-51　高斯低通滤波的结果

2. 非线性平滑滤波器

中值滤波器是最常用的中值滤波器。它是一种邻域运算，类似于卷积，但计算的不是加权求和，而是把邻域中的像素按灰度级进行排序，然后选择该组的中间值作为输出像素值。具体步骤是：

（1）将模板在图像中漫游，并将模板中心与图像中某个像素的位置重合。

（2）读取模板下各对应像素的灰度值。

（3）将这些灰度值从小到大排成一列。

（4）找出这些值里排在中间的一个。

（5）将这个中间值赋给对应模板中心位置的像素。

由此可以看出，中值滤波器的主要功能就是让与周围像素灰度值的差比较大的像素改取与周围的像素值接近的值，从而可以消除孤立的噪声点。

例程 8-15 调用 B = medfilt2(A,[m n])来实现中值滤波器。

例程 8-15

```
%下面仍以 saturn.tif 为例，加入椒盐噪声，并采用中值过滤器
I = imread('saturn.tif');
J = imnoise(I,'salt & pepper',0.02);
K1 = medfilt2(J, [3,3]);
figure,imshow(K1)
```



函数 K = medfilt2(A,[m n])使用 m × n 的模板来完成中值滤波，默认模板是 3 × 3 的。在以上代码的执行过程中，你会发现中值滤波的计算比卷积的运算要慢，这是由于它需要对邻域中的所有像素按灰度级排序。

采用默认的模板做中值滤波的结果如图 8-52 所示。

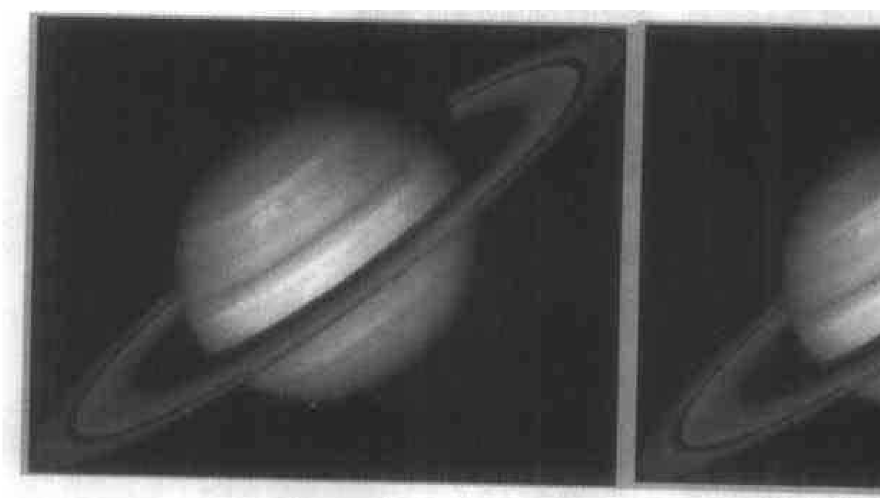


图 8-52　medfilt2(J,[3,3])处理结果



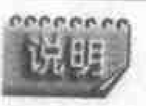
图 8-53　ordfilt2(J,5,ones(3,3))处理结果

例程 8-16 调用 B = ordfilt2(A,order,domain)函数来实现中值滤波。

例程 8-16

```
I = imread('saturn.tif');
J = imnoise(I,'salt & pepper',0.02);
K1= ordfilt2(J,5,ones(3,3));
figure,imshow(K1)
```

以上所采用的滤波器也是 3×3 的中值滤波器，效果如图 8-53 所示。



在函数的 3 个参数中，domain 表示邻域的形状，order 表示所取的像素在邻域像素的排序中所处的位置。当 domain 是全 1 的 3 × 3 方阵时，邻域内共有 9 个元素，则排在中间位置的元素为第 5 个。

由以上处理结果可以看出，中值滤波器不像均值滤波器那样，它在衰减噪声的同时不会使图像的边界模糊，这也是中值滤波器受欢迎的主要原因。中值滤波器去噪声的效果依赖于两个要素：邻域的空间范围，中值计算中所涉及的像素数。一般来说，小于中值滤波器面积一半的亮或暗的物体基本上会被滤掉，而较大的物体则几乎会原封不动地保存下

来。因此中值滤波器的空间尺寸必须根据手中的问题来进行调整。较简单的模板是 N×N 的方形（这里 N 常是奇数），计算时用到所有 N^2 个像素点。另外，我们也可以使用稀疏分布的模板来节省时间。

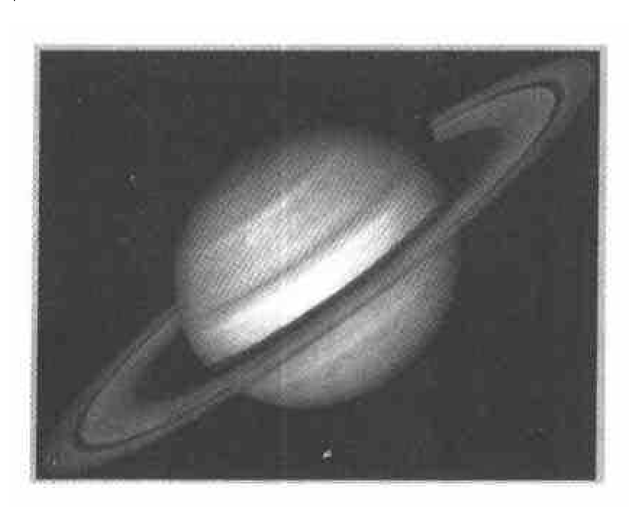


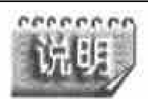
图 8-54　ordfilt2(J,5,domain)处理结果

例程 8-17 使用稀疏分布的模板实现中值滤波。

例程 8-17

```
I = imread('saturn.tif');
J = imnoise(I,'salt & pepper',0.02);
domain=[0 0 1 0 0;
        0 0 1 0 0;
        1 1 1 1 1;
        0 0 1 0 0;
        0 0 1 0 0];
K1= ordfilt2(J,5,domain);
figure,imshow(K1)
```

结果如图 8-54 所示。



尽管 domain 是 5×5 的邻域，但由于它是稀疏矩阵，其中非 0 元素只有 9 个，所以在排序完成之后排在中间位置的元素仍为第 5 个。

类似的稀疏矩阵还有很多，比如：

```
domain1=[1 0 0 0 1;
         0 1 0 1 0;
         0 0 1 0 0;
         0 1 0 1 0;
         1 0 0 0 1];
domain2=[0 0 1 0 0;
         0 1 1 1 0;
```

```
1 1 1 1 1;
0 1 1 1 0;
0 0 1 0 0];
```

当然对应于 domain1，中值应取第 5 个像素；而对应于 domain2，中值应取第 7 个像素大的模板；对于其他不同的模板，中值也要依情况而定。计算中值时所使用的像素数的增加跟去噪时的效果之间的关系是非线性的。如果手中的问题需要用到大尺度的中值滤波器，那么用稀疏分布的中值模板或许能得到更令人满意的效果。

中值滤波只是排序统计滤波中的一种。如果邻域中的输入像素已经排好序了，那么中值代表的是第 50 个百分点的数值。其他百分点的数值也可以用。第 0 个百分点可用于最小值滤波器，它用来检测图像中最暗的点；第 100 个百分点可用于最大值滤波器，它用来检测图像中最亮的点；或利用不是第 50 个百分点来滤波趋向于使图像变暗或变亮。在 MATLAB 中这几种滤波器都可以用 B = ordfilt2(A,order,domain)函数来实现，分别举例如下：

B = ordfilt2(A,1,ones(3,3))实现 3×3 的最小值滤波器，因为它取全 1 的模板中排在最小位置处的那个像素。

B = ordfilt2(A,9,ones(3,3))实现 3×3 的最大值滤波器，因为它取全 1 的模板中排在最大位置处的那个像素。

B = ordfilt2(A,1,[0 1 0; 1 0 1; 0 1 0]) 的输出是每个像素的东、南、西、北四个方向相邻像素灰度的最小值，因为它取四邻域的模板中排在最小位置处的那个像素。

8.2.3 锐化滤波器

1. 线性锐化滤波器

线性高通滤波器是最常用的线性锐化滤波器。这种滤波器的中心系数都是正的，而周围的系数都是负的。对 3×3 的模板来说，典型的系数取值是：

[-1 -1 -1; -1 8 -1; -1 -1 -1]

事实上这是拉普拉斯算子，所有系数之和为 0。当这样的模板放在图像中灰度值是常数或变化很小的区域时，其输出为 0 或很小。这个滤波器将原图像中的零频率分量去除了，也就是将输出图像的平均灰度值变为 0，这样就会有一部分像素灰度值小于 0。在图像处理中我们一般只考虑正的灰度值，所以还要将输出图像的灰度值范围通过尺度变换变回到所要求的范围。

在 MATLAB 中可通过调用 filter2 函数和 fspecial 函数来实现，代码如下：

```
I = imread('saturn.tif');
h = fspecial('laplacian');
I2 = filter2(h,I);
imshow(I)
figure,imshow(I2)
```



在 fspecial('laplacian',alpha) 函数中，alpha 控制 Laplacian 算子的形状，取值范围为[0，1]，默认值为 0.2。

原始图像和处理的效果分别如图8-55和图8-56所示。

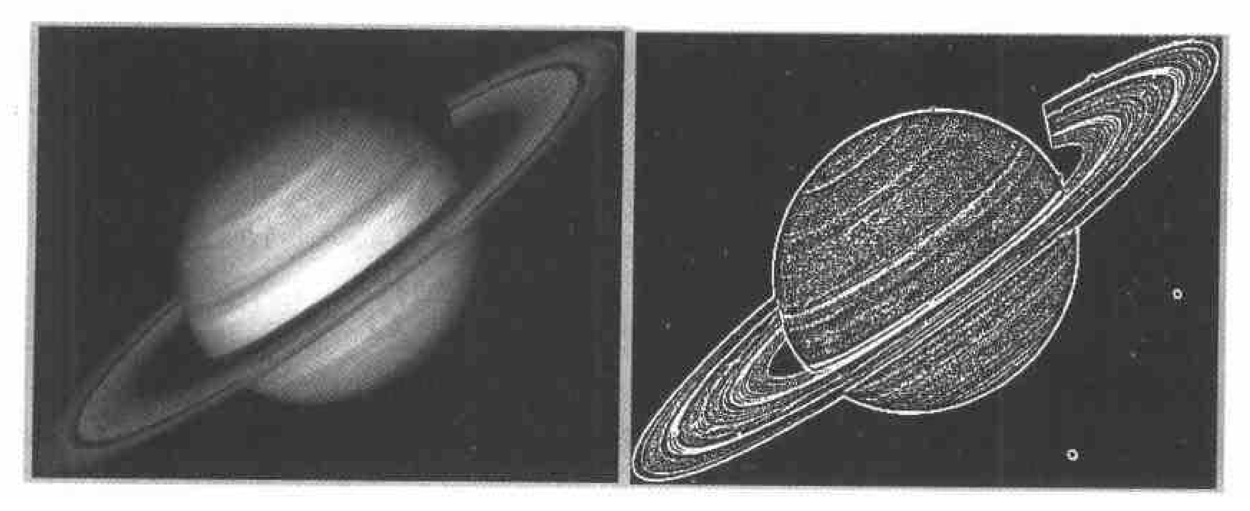


图 8-55　原始图像　　　　图 8-56　拉普拉斯算子处理的效果

采用“unsharp”算子实现对比度增强滤波器，代码如下：

```
I = imread('saturn.tif');
h = fspecial('unsharp',0.5);
I2 = filter2(h,I)/255;
figure,imshow(I2)
```



在 h = fspecial('unsharp', alpha)函数中，alpha 控制滤波器的形状，取值范围为[0，1]，默认值为 0.2。

处理效果如图8-57所示。关于锐化滤波器在后面的边缘检测中还要详细介绍，这里为了体系结果的完整，只做简单介绍。

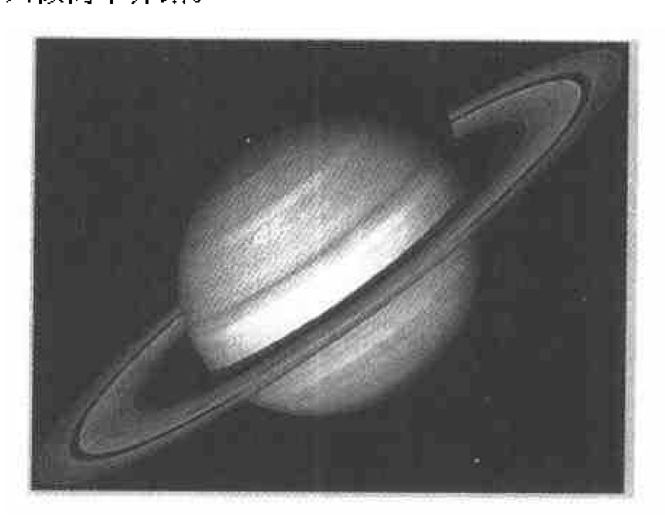


图 8-57　对比度增强滤波器处理的效果

2. 非线性锐化滤波器

邻域平均可以模糊图像，因为平均对应积分，所以利用微分可以锐化图像。图像处理

中最常用的微分方法是利用梯度。对一个连续函数 $f(x,y)$，其梯度是一个矢量（需要用两个模板分别沿x和y方向计算）：

$$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]^T$$

其模（以2为模，对应欧氏距离）为：

$$\left|\nabla f_{(2)}\right| = mag(\nabla f)\left[\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2\right]^{\frac{1}{2}}$$

在使用中为了计算简便，也可不用上述对应欧氏距离的以2为模的方法组合两个模板的输出。有一种简单的方法是利用城区距离（以1为模），即：

$$\left|\nabla f_{(1)}\right| = \left|\frac{\partial f}{\partial x}\right| + \left|\frac{\partial f}{\partial y}\right|$$

另一种简单方法是利用棋盘距离（以∞为模），即：

$$\left|\nabla f_{(\infty)}\right| = \max\left\{\left|\frac{\partial f}{\partial x}\right|, \left|\frac{\partial f}{\partial y}\right|\right\}$$

上述这些组合的方法本身都是非线性的。常用的空域微分算子有sobel算子、prewitt算子、高斯—拉普拉斯算子等。

例程8-18对几种边缘增强算子的效果进行比较。

例程 8-18

```
I1=imread('blood1.tif');
figure,imshow(I1)
h1=fspecial('sobel');
I2=filter2(h1,I1);
figure,imshow(I2)
I3=conv2(I1,h1);
figure,imshow(I3)
h2=fspecial('prewitt');
I4=filter2(h2,I1);
figure,imshow(I4)
h3=fspecial('log');
I5=filter2(h3,I1);
figure,imshow(I5)
```

结果如图8-58、图8-59、图8-60、图8-61和图8-62所示。

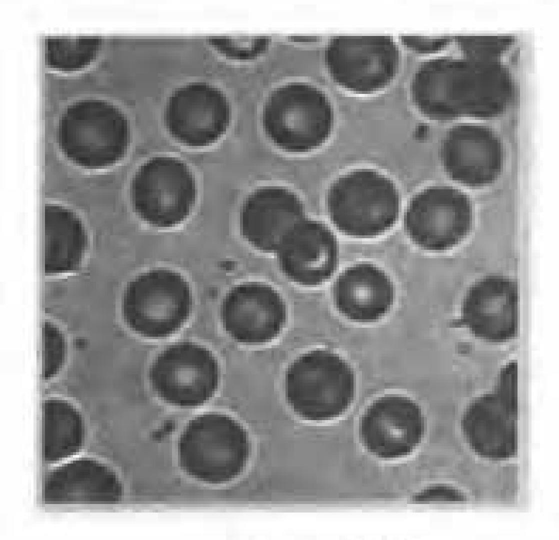


图 8-58　原始图像

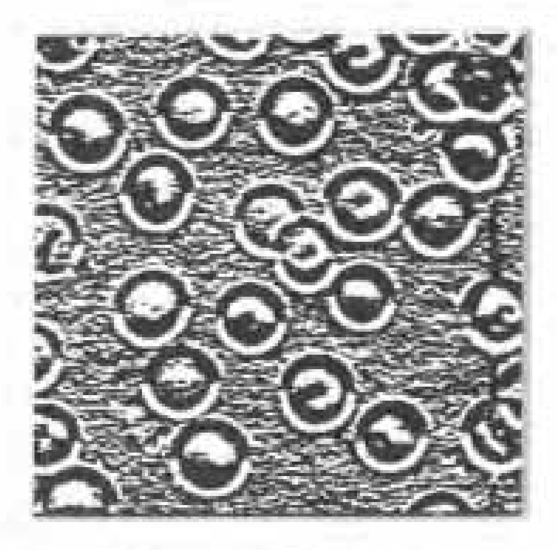


图 8-59　sobel 卷积



图 8-60　sobel 滤波



图 8-61　prewitt 滤波

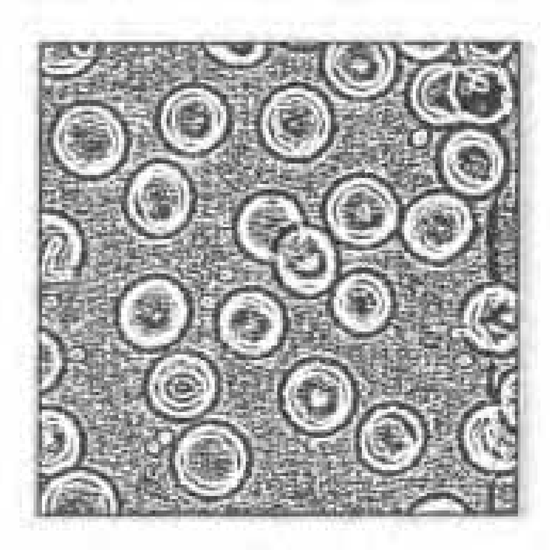


图 8-62　log 滤波



8.3 频域增强

卷积理论是频域技术的基础。设函数 $f(x,y)$ 与线性位不变算子 $h(x,y)$ 的卷积结果是 $g(x,y)$，即

$$g(x,y)=h(x,y)\times f(x,y)$$

那么根据卷积定理在频域有：

$$G(u,v)=H(u,v)F(u,v)$$

其中 $G(u,v)$、$H(u,v)$、$F(u,v)$ 分别是 $g(x,y)$、$h(x,y)$、$f(x,y)$ 的傅里叶变换。

频域增强的主要步骤是：

（1）计算所需增强图的傅里叶变换。

（2）将其与一个（根据需要设计的）转移函数相乘。

（3）再将结果进行傅里叶反变换以得到增强的图。

频域增强方法有两个关键：

- 将图像从空域转换到频域所需的变换（设用 T 表示）及将图像从频域空间转换回空域所需的变换（设用 T^{-1} 表示）。
- 在频域空间对图像进行增强加工的操作（设仍用 EH 表示），此时对应的增强可表示为：

$$g(x,y)=T^{-1}\{EH[T[f(x,y)]]\} \qquad (8\text{-}10)$$

常用的频域增强方法有低通滤波和高通滤波。以下分别介绍在 MATLAB 中如何实现。

8.3.1 低通滤波

图像的能量大部分集中在幅度谱的低频和中频度，而图像的边缘和噪声对应于高频部分。因此能降低高频成分幅度的滤波器就能减弱噪声的影响。

Butterworth 低通滤波器是一种物理上可以实现的低通滤波器。n 阶，截断频率为 d0 的 Butterworth 低通滤波器的转移函数为：

$$H(u,v)=\frac{1}{1+[\mathrm{d}(u,v)/\mathrm{d}0]^{2n}}$$

例程 8-19 实现 Butterworth 低通滤波器。

例程 8-19

```
I1=imread('Saturn.tif');
figure,imshow(I1)
I2=imnoise(I1,'salt & pepper');
figure,imshow(I2)
f=double(I2);
%傅里叶变换
g=fft2(f);
%转换数据矩阵
g=fftshift(g);
[N1,N2]=size(g);
n=2;
d0=50;
n1=fix(N1/2);
n2=fix(N2/2);
for i=1:N1
    for j=1:N2
        d=sqrt((i-n1)^2+(j-n2)^2);
       %计算 Butterworth 低通转换函数
h=1/(1+0.414*(d/d0)^(2*n));
        result(i,j)=h*g(i,j);
    end
end

result =ifftshift(result);
X2=ifft2(result);
X3=uint8(real(X2));
figure,imshow(X3)
```

原图和处理结果如图 8-63 和图 8-64 所示。



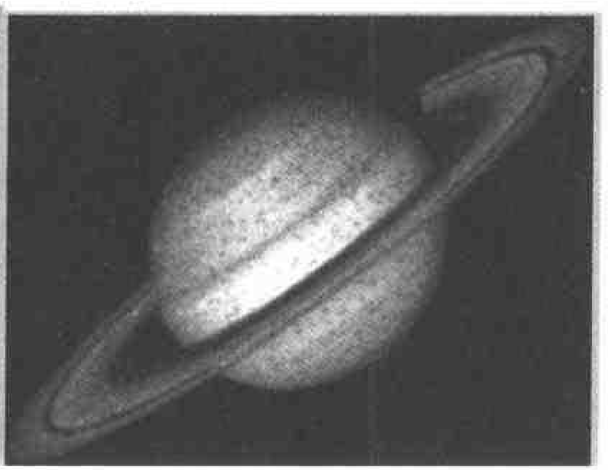


图 8-63　加噪图像

图 8-64　Butterworth 低通滤波器去噪图像



8.3.2 高通滤波

高通滤波也称高频滤波器，它的频值在 0 频率处为单位 1，随着频率的增长，传递函数的值逐渐增加；当频率增加到一定值之后传递函数的值通常又回到 0 值或者降低到某个大于 1 的值。在前一种情况下，高频增强滤波器实际上是一种带通滤波器，只不过规定 0 频率处的增益为单位 1。

实际应用中，为了减少图像中面积大且缓慢变化的成分的对比度，有时让 0 频率处的增益小于单位 1 更合适。如果传递函数通过原点，则可以称为 laplacian 滤波器。

n 阶截断频率为 d0 的 Butterworth 高通滤波器的转移函数为:

$$H(u,v)=\frac{1}{1+\left[\mathrm{d}0/\mathrm{d}(u,v)\right]^{2n}}$$

例程 8-20 实现 Butterworth 高通滤波器。

例程 8-20

```
I1=imread('blood1.tif');
figure,imshow(I1)
f=double(I1);
g=fft2(f);%傅里叶变换
g=fftshift(g);%转换数据矩阵
[N1,N2]=size(g);
n=2;
d0=5;
n1=fix(N1/2);
n2=fix(N2/2);
for i=1:N1
    for j=1:N2
      d=sqrt((i-n1)^2+(j-n2)^2);
      if d==0
          h=0;
      else
          h=1/(1+(d0/d)^(2*n));
       end
      result(i,j)=h*g(i,j);
    end
end
result =ifftshift(result);
X2=ifft2(result);
X3=uint8(real(X2));
figure,imshow(X3)
```

原始图像和处理结果如图 8-65 和图 8-66 所示。

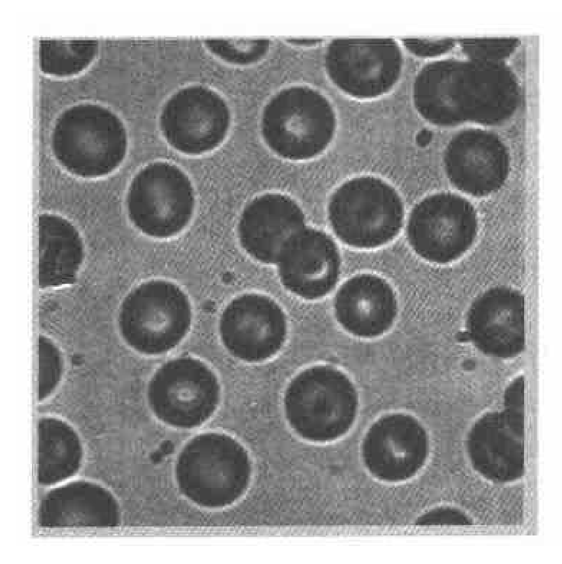


图 8-65　原始图像

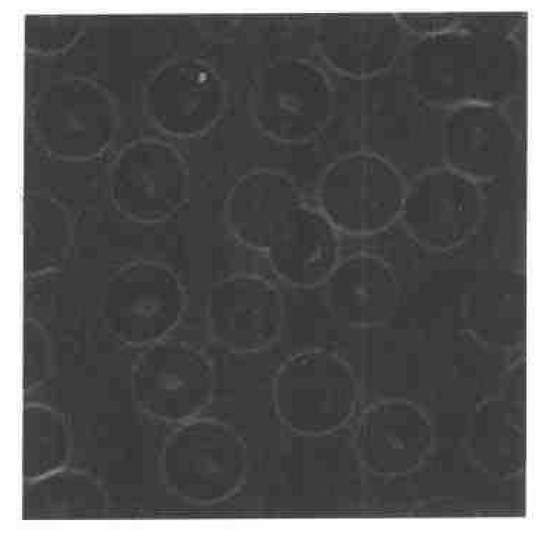


图 8-66　Butterworth 高通滤波图像

从所示的图 8-66 高通滤波图像可见，图像很昏暗，很多细节都看不清了。这是由于图像的大部分能量集中在低频区域，而高通滤波使图中各区域的边界得到较明显的增强的同时滤掉了低频分量，使图中原来比较平滑区域内部的灰度动态范围被压缩，因而整幅图变得比较昏暗。

第 9 章　四叉树分解与边缘检测

四叉树分解是一种基于均匀性检测的图像分割方法。边缘检测利用边缘增强算子，突出图像中的局部边缘，并提取边缘点集。这两种技术在工程实践中应用比较广泛。MATLAB 图像处理工具箱同样为这两种图像处理方法提供了丰富的函数，在这一章我们将对其进行介绍，并列举大量的工程实例。

本章主要内容：

- 四叉树分解
- 边缘检测

9.1　四叉树分解

四叉树分解是一种基于均匀性检测的图像分割方法，在图像分析和图像压缩中应用广泛。本节介绍四叉树分解的基本原理，应用 MATLAB 图像处理工具箱中的函数来实现四叉树分解。

9.1.1　四叉树分解的基本原理及 MATLAB 工具箱函数

四叉树分解将原始图像逐步细分成小块，操作的目标是将具有一致性的像素分到同一个小块中。通常这些小块都是方块，只有少数情况分成长方形。因此当图像是方形，且像素点的个数是 2 的整数次幂时，四叉树分解法最适用。这里所说的一致性通常指方块内部各像素点的灰度值的范围不超过某个阈值，也就是各像素点的灰度值的接近程度满足要求。

进行四叉树分解的具体过程是：将方形的原始图像分成四个相同大小的方块，判断每个方块是否满足一致性标准；如果满足就不再继续分裂，如果不满足就再细分成四个方块，并对细分得到的方块应用一致性检验。这个迭代重复的过程直到所有的方块都满足一致性标准才停止。最后，四叉树分解的结果可能包含多种不同尺寸的方块。图 9-1 是一幅图像四叉树分解的结果，从中可以看到各种不同尺寸的方块：有的是第二次分裂得到的，尺寸是整幅图的 1/16；有的是第三次分裂得到的，尺寸是整幅图的 1/64。这幅图总共进行了 6 次分裂，得到最终的结果。

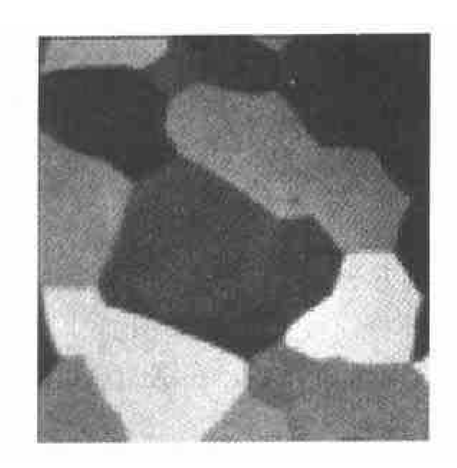
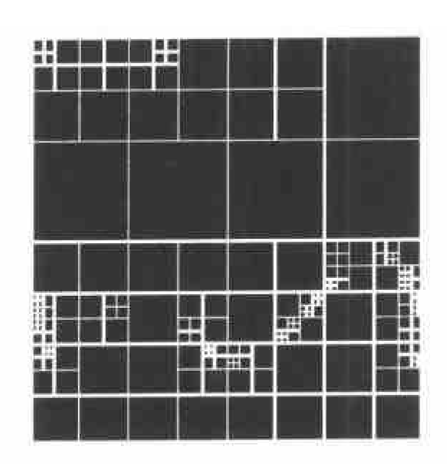



图 9-1　图像的四叉树分解

用于四叉树分解的函数有：qtdecomp 函数、qtgetblk 函数、qtsetblk 函数。以下分别介绍它们的语法及用法。

1．qtdecomp 函数

qtdecomp 函数的语法格式如下：

```
S = qtdecomp(I)
S = qtdecomp(I,threshold)
S = qtdecomp(I,threshold,mindim)
S = qtdecomp(I,threshold,[mindim maxdim])
S = qtdecomp(I,fun)
S = qtdecomp(I,fun,P1,P2···)
```

S = qtdecomp(I) 对灰度图像 I 进行四叉树分解，它递归地将图像分成子块，直到同一子块中的所有元素值都相等。函数将四叉树结构返回到稀疏矩阵 S 中；如果其中的元素 $S(k,m)$ 不等于 0，则 (k,m) 是分解结果中一个方块的左上角，而 $S(k,m)$ 的值就是其所属方块的尺寸。

S = qtdecomp(I,threshold)根据阈值 threshold 对原始图像 I 进行四叉树分解。分解过程直到每个方块中最大值元素与最小值元素之间的差值不大于阈值才结束。无论 I 是 uint8 型还是 uint16 型，threshold 的取值范围都是[0,1]；当 I 是 uint8 型时，要将阈值乘以 255 作为实际使用的值；当 I 是 uint16 型时，要将阈值乘以 65535 作为实际使用的值。

S = qtdecomp(I,threshold,mindim)产生的方块尺寸不小于 mindim，即使最后的结果中有一些方块不满足一致性条件。

S = qtdecomp(I,threshold,[mindim maxdim]) 产生的方块尺寸不小于 mindim，不大于 maxdim。尺寸大于 maxdim 的方块即使已经满足了一致性条件也要继续分解。mindim 和 maxdim 都必须是 2 的整数次幂。

S = qtdecomp(I,fun)利用函数 fun 来确定方块是否需要继续分裂。qtdecomp 函数对 $m\times m\times k$ 堆栈中尺寸为 $m\times m$ 的所有当前方块调用 fun 函数，其中 k 是 $m\times m$ 方块的个数。



fun 函数返回 k 维矢量，每个元素只取 0 或 1。其中 1 表示对应的方块需要继续分解，0 表示对应的方块不需要继续分解，例如，如果 k(3)=0，那么第三个 m×m 方块不需要继续分解。fun 是一个字符串，包含函数名、表达式或联机函数。

S = qtdecomp(I,fun,P1,P2,…)将 P1,P2,…作为参数传递给 fun。

数据类型说明：在不包含函数的语法中，输入图像可以是 uint8 型、uint16 型或 double 型。在包含函数的语法中，输入图像可以是该函数支持的任何类型。输出矩阵总是 sparse 型的。

当qtdecomp函数处理的图像尺寸是2的整数次幂（128×128或256×256）时，这些图像最小可以分成1×1的方块。当所处理的图像尺寸不是2的整数次幂时，在有些点上方块就不能继续再分解。例如，如果图像是96×96的，则可分成48×48，然后分成24×24，然后分成12×12，然后分成6×6，最后分成3×3，这时就不能分成更小的方块了。处理这种图像时，必须将mindim设置为3（或2^3）。如果用到包含了函数的语法，则在不能再分的方块处函数不许返回0。

例程9-1进行四叉树分解。

例程 9-1

```
%假设下面的 I 是一幅灰度图像，取阈值 0.5，进行四叉树分解
I = [1    1    1    1    2    3    6    6
1    1    2    1    4    5    6    8
1    1    1    1    10   15   7    7
1    1    1    1    20   25   7    7
20   22   20   22   1    2    3    4
20   22   22   20   5    6    7    8
20   22   20   20   9    10   11   12
22   22   20   20   13   14   15   16];
S = qtdecomp(I, 0.5)
```

输出结果：

```
S =
   (1,1)        2
   (3,1)        2
   (5,1)        1
   (6,1)        1
   (7,1)        1
   (8,1)        1
   (5,2)        1
   (6,2)        1
   (7,2)        1
   (8,2)        1
   (1,3)        1
   (2,3)        1
   (3,3)        2
```

```
   (5,3)        1
   (6,3)        1
   (7,3)        2
   (1,4)        1
   (2,4)        1
   (5,4)        1
   (6,4)        1
   (1,5)        1
   (2,5)        1
   (3,5)        1
   (4,5)        1
   (5,5)        1
   (6,5)        1
   (7,5)        1
   (8,5)        1
   (1,6)        1
   (2,6)        1
   (3,6)        1
   (4,6)        1
   (5,6)        1
   (6,6)        1
   (7,6)        1
   (8,6)        1
   (1,7)        1
   (2,7)        1
   (3,7)        2
   (5,7)        1
   (6,7)        1
   (7,7)        1
   (8,7)        1
   (1,8)        1
   (2,8)        1
   (5,8)        1
   (6,8)        1
   (7,8)        1
   (8,8)        1
full(S)
ans =
     2     0     1     1     1     1     1     1
     0     0     1     1     1     1     1     1
     2     0     2     0     1     1     2     0
     0     0     0     0     1     1     0     0
     1     1     1     1     1     1     1     1
     1     1     1     1     1     1     1     1
     1     1     2     0     1     1     1     1
     1     1     0     0     1     1     1     1
```



2．qtgetblk 函数

qtgetblk 函数的语法格式如下：

```
[vals,r,c] = qtgetblk(I,S,dim)
[vals,idx] = qtgetblk(I,S,dim)
```

[vals,r,c] = qtgetblk(I,S,dim) 返回图像 I 的四叉树分解中 dim×dim 子块的值，存放在数组 vals 中。S 为由 qtdecomp 函数返回的稀疏矩阵，包含了四叉树结构。vals 是一个 dim×dim×k 维的数组；k 是四叉树分解中 dim×dim 维方块的个数；如果四叉树分解中没有指定尺寸的子块，则返回空矩阵。r、c 分别存放行、列坐标的矢量，对应着各个子块的左上角。

[vals,idx] = qtgetblk(I,S,dim) 返回各个子块的左上角的线性索引，存放在矢量 idx 中。

数据类型要求：I 可以是 uint8 型、uint16 型或 double 型。S 是 sparse 型的。

数组 vals 中方块的顺序对应着 I 中方块的列顺序。例如，如果 vals 是 4×4×2 的，vals(:,:,1) 包含的值就来自于 I 中的第一个 4×4 方块，vals(:,:,2) 包含的值就来自于 I 中的第二个 4×4 方块。

例程 9-2 调用 qtgetblk 函数。

例程 9-2

```
%S 由例程 9-1 得到
[vals,r,c] = qtgetblk(I,S,2)
```

输出结果：

```
vals(:,:,1) =
              1     1
              1     1
vals(:,:,2) =
1     1
              1     1
vals(:,:,3) =
              1     1
              1     1
vals(:,:,4) =
             20    20
             20    20
vals(:,:,5) =
              7     7
              7     7
r =
     1
     3
     3
     7
```

```
        3
    c =
        1
        1
        3
        3
        7
```

3．qtsetblk 函数

qtsetblk 函数的语法格式如下：

```
J = qtsetblk(I,S,dim,vals)
```

J = qtsetblk(I,S,dim,vals) 将图像 I 的四叉树分解中每个 dim×dim 维子块的值全部替换为 vals 中 dim×dim 维的方块，S 为由 qtdecomp 函数返回的稀疏矩阵，包含了四叉树结构。vals 是一个 dim×dim×k 维的数组；k 是四叉树分解中 dim×dim 维方块的个数。

数据类型要求：I 可以是 uint8 型、uint16 型或 double 型。S 是 sparse 型的。

数组 vals 中方块的顺序必须与 I 中方块的列顺序相对应。例如，如果 vals 是 4×4×2 的，vals(:,:,1) 包含的值就用来替换 I 中的第一个 4×4 方块，vals(:,:,2) 包含的值就用来替换 I 中的第二个 4×4 方块。

例程9-3调用qtsetblk函数。

例程 9-3

```
newvals= cat(3,zeros(2),ones(2),2*ones(2),3*ones(2),4*ones(2))
J = qtsetblk(I,S,2,newvals)
```

输出结果：

```
newvals(:,:,1) =
        0   0
        0   0
newvals(:,:,2) =
        1   1
        1   1
newvals(:,:,3) =
        2   2
        2   2
newvals(:,:,4) =
        3   3
        3   3
newvals(:,:,5) =
        4   4
        4   4
J =
        0   0   1   1   2   3   6   6
        0   0   2   1   4   5   6   8
```



```
1    1    2    2    10   15   4    4
1    1    2    2    20   25   4    4
20   22   20   22   1    2    3    4
20   22   22   20   5    6    7    8
20   22   3    3    9    10   11   12
22   22   3    3    13   14   15   16
```

9.1.2　应用四叉树分解

例程 9-4 以 rice.tif 为例，进行四叉树分解，并显示稀疏矩阵的结果。

例程 9-4

```
%首先，显示原始图像
I1=imread('rice.tif');
figure,imshow(I1)
%原始图像如图 9-2 所示
%然后，选取阈值为 0.2，对原始图像进行四叉树分解，并以灰度图的形式显示分解所得的稀疏矩阵。
S = qtdecomp(I1,0.2);
S2=full(S);
figure,imshow(S2)
%稀疏矩阵如图 9-3 所示
%最后，通过查看 S2 来了解四叉树分解的具体结果
[vals,r,c] = qtgetblk(I1,S2,2);
size(vals)
[vals2,r,c] = qtgetblk(I1,S2,4);
size(vals2)
[vals3,r,c] = qtgetblk(I1,S2,8);
size(vals3)
[vals4,r,c] = qtgetblk(I1,S2,16);
size(vals4)
[vals5,r,c] = qtgetblk(I1,S2,32);
size(vals5)
[vals6,r,c] = qtgetblk(I1,S2,1);
size(vals6)
```

输出结果：

```
ans = 2      2     4166
ans = 4      4     1540
ans = 8      8      317
ans = 16     16      12
ans = 32     32       0
ans = 1      1      872
```

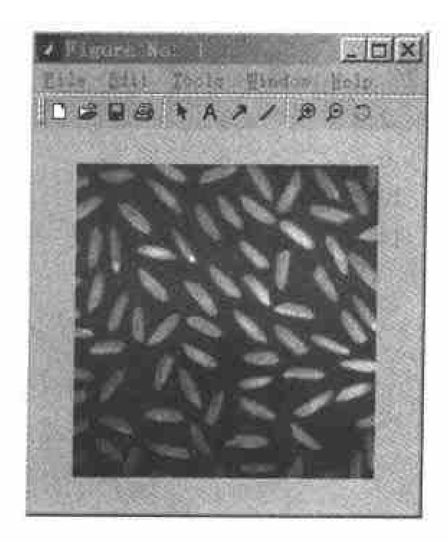


图 9-2 rice.tif 原始图像



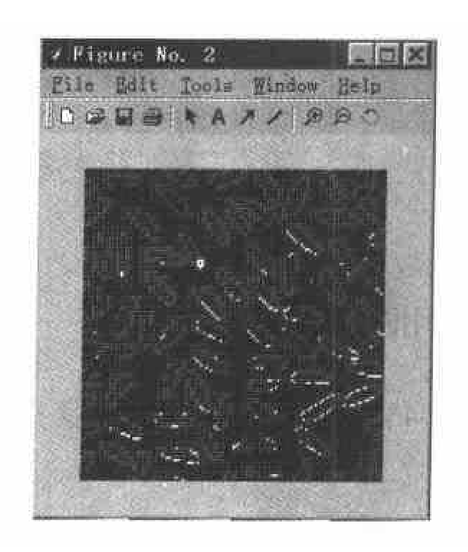


图 9-3 阈值为 0.2 的稀疏矩阵

以上结果说明在 0.2 的阈值下，四叉树分解的最小方块是 1×1 的，最大方块是 32×32 的。当阈值选取不同时，四叉树分解所得的稀疏矩阵会有差别，读者可以根据应用需要，选取合适的阈值。

9.2 边 缘 检 测

边缘检测在图像处理与计算机视觉中占有特殊的位置，它是底层视觉处理中最重要的环节之一。

9.2.1 边缘检测的基本原理及处理函数

两个具有不同灰度值的相邻区域之间总存在边缘，边缘是灰度值不连续的结果，这种不连续性通常可以利用求导数的方法方便地检测到。一般常用一阶导数和二阶导数来检测边缘。

边缘检测的基本思想是首先利用边缘增强算子，突出图像中的局部边缘，然后定义像素的"边缘强度"，通过设置门限的方法提取边缘点集。

常用的边缘检测算子有 Robert 算子、Sobel 算子、Prewitt 算子、LOG 算子和 Canny 算子，以下分别进行介绍。

1．Robert 算子

Robert 算子是一种利用局部差分算子寻找边缘的算子，其模板如图 9-4 所示。Robert 算子对具有陡峭的低噪声的图像效果较好。

1	0
0	-1

0	1
-1	0

图 9-4 Robert 算子

在 MATLAB 中可以由 edge 函数实现，语法格式如下：

```
BW = edge(I,'roberts')
BW = edge(I,'roberts',thresh)
[BW,thresh] = edge(I,'roberts',…)
```

BW = edge(I,'roberts') 自动选择阈值用 Robert 算子进行边缘检测。

BW = edge(I,'roberts',thresh)根据所指定的敏感度阈值 thresh 用 Robert 算子进行边缘检测，它忽略了所有小于阈值的边缘。当 thresh 为空（[]）时，自动选择阈值。

[BW,thresh] = edge(I,'roberts',…) 返回阈值。

edge 函数对灰度图像 I 进行边缘检测，返回与 I 同样大的二值图像 BW，其中 1 表示 I 中的边缘，0 表示非边缘。I 是 uint8 型、uint16 型，或 double 型，BW 是 uint8 型的。

2．Sobel 算子

Sobel 算子的两个卷积计算核如图 9-5 所示，图像中的每个点都用这两个核做卷积，第一个核对垂直边缘响应最大，第二个核对水平边缘响应最大。两个卷积的最大值作为该点的输出值，运算结果是一幅边缘幅度图像。Sobel 算子对灰度渐变和噪声较多的图像处理得较好。

1	2	1
0	0	0
-1	-2	-1

1	0	-1
2	0	-2
1	0	-1

图 9-5　Sobel 算子

edge 函数实现的语法格式如下：

```
BW = edge(I,'sobel')
BW = edge(I,'sobel',thresh)
BW = edge(I,'sobel',thresh,direction)
[BW,thresh] = edge(I,'sobel',…)
```

BW = edge(I,'sobel') 自动选择阈值用 Sobel 算子进行边缘检测。

BW = edge(I,'sobel',thresh) 根据所指定的敏感度阈值 thresh，用 Sobel 算子进行边缘检测，它忽略了所有小于阈值的边缘。当 thresh 为空（[]）时，自动选择阈值。

BW = edge(I,'sobel',thresh,direction) 根据所指定的敏感度阈值 thresh，在所指定的方向 direction 上，用 Sobel 算子进行边缘检测。direction 可取的字符串值为'horizontal'（水平方向）、'vertical'（垂直方向）或'both'（两个方向）。

[BW,thresh] = edge(I,'sobel',…) 返回阈值.

3．Prewitt 算子

Prewitt 算子的两个卷积计算核如图 9-6 所示。与使用 Sobel 算子的方法一样，图像中的每个点都用这两个核做卷积，取最大值作为输出 Prewitt 算子也产生一幅边缘幅度图像，也是对灰度渐变和噪声较多的图像处理得较好。

1	0	-1
1	0	-1
1	0	-1

1	1	1
0	0	0
-1	-1	-1

图 9-6 Prewitt 算子

edge 函数实现的语法格式如下：

```
BW = edge(I,'prewitt')
BW = edge(I,'prewitt',thresh)
BW = edge(I,'prewitt',thresh,direction)
[BW,thresh] = edge(I,'prewitt',…)
```

BW = edge(I,'prewitt') 自动选择阈值用 Prewitt 算子进行边缘检测。

BW = edge(I,'prewitt',thresh) 根据所指定的敏感度阈值 thresh，用 Prewitt 算子进行边缘检测，忽略了所有小于阈值的边缘。当 thresh 为空（[]）时，自动选择阈值。

BW = edge(I,'prewitt',thresh,direction) 根据所指定的敏感度阈值 thresh，在所指定的方向 direction 上，用 Prewitt 算子进行边缘检测。direction 可取的字符串值为'horizontal'（水平方向）、'vertical'（垂直方向）或'both'（两个方向）。

[BW,thresh] = edge(I,'prewitt',…) 返回阈值。

4．LOG 算子

拉普拉斯算子的卷积核如图 9-7 所示，它是一个二阶算子，将在边缘处产生一个陡峭的零交叉。拉普拉斯算子是一个线性的、移不变的算子，它的传递函数在频域空间的原点是零，因此经拉普拉斯滤波过的图像具有零平均灰度。LOG 算子先用高斯低通滤波器将图像进行预先平滑，然后用拉普拉斯算子找出图像中的陡峭边缘，最后用零灰度值进行二值化产生闭合的、连通的轮廓，消除了所有内部点。

0	-1	0
-1	4	-1
0	-1	0

-1	-1	-1
-1	8	-1
-1	-1	-1

图 9-7 拉普拉斯算子

edge 函数实现的语法格式如下：

```
BW = edge(I,'log')
BW = edge(I,'log',thresh)
BW = edge(I,'log',thresh,sigma)
[BW,threshold] = edge(I,'log',…)
```

BW = edge(I,'log') 自动选择阈值用 LOG 算子进行边缘检测。

BW = edge(I,'log',thresh) 根据所指定的敏感度阈值 thresh，用 LOG 算子进行边缘检测，它忽略了所有小于阈值的边缘。当 thresh 为空（[]）时，自动选择阈值。当指定 thresh 为 0 时，输出图像具有闭合的轮廓，因为其中包含了输入图像中的所有零交叉点。

BW = edge(I,'log',thresh,sigma) 根据所指定的敏感度阈值 thresh 和标准偏差 sigma，用 LOG 算子进行边缘检测，默认时 sigma 等于 2，滤波器是 n×n 维的，其中

$$n=ceil(sigma\times 3)\times 2+1$$

[BW,threshold] = edge(I,'log',…) 返回阈值。

5．Canny 算子

Canny 算子检测边缘的方法是寻找图像梯度的局部极大值，梯度是用高斯滤波器的导数计算的。Canny 方法使用两个阈值来分别检测强边缘和弱边缘，而且仅当弱边缘与强边缘相连时，弱边缘才会包含在输出中。因此，此方法不容易受噪声的干扰，能够检测到真正的弱边缘。

edge 函数实现的语法格式如下：

```
BW = edge(I,'canny')
BW = edge(I,'canny',thresh)
BW = edge(I,'canny',thresh,sigma)
[BW,threshold] = edge(I,'canny',…)
```

BW = edge(I,'canny') 自动选择阈值用 Canny 算子进行边缘检测。

BW = edge(I,'canny',thresh) 根据所指定的敏感度阈值 thresh，用 LOG 算子进行边缘检测。thresh 是一个含两个元素的矢量，第一个元素是低阈值，第二个元素是高阈值；如果只给 thresh 指定一个值，则此值作为高阈值，而 0.4×thresh 作为低阈值；当 thresh 为空（[]）时，自动选择低阈值和高阈值。

BW = edge(I,'canny',thresh,sigma) 根据所指定的敏感度阈值 thresh 和标准偏差 sigma，用 Canny 算子进行边缘检测，默认时 sigma 等于 1，滤波器的尺寸 sigma 根据自动选择。

[BW,threshold] = edge(I,'canny',…) 返回含两个元素的阈值矢量。

6．零交叉方法

零交叉方法先用指定的滤波器对图像进行滤波，然后寻找零交叉点作为边缘。edge 函数实现的语法格式如下：

```
BW = edge(I,'zerocross',thresh,h)
[BW,thresh] = edge(I,'zerocross',…)
```

BW = edge(I,'zerocross',thresh,h) 根据所指定的敏感度阈值 thresh 和滤波器 h，用零交叉方法进行边缘检测。当 thresh 为空（[]）时，自动选择阈值；当指定 thresh 为 0 时，输出图像具有闭合的轮廓，因为其中包含了输入图像中的所有零交叉点。

[BW,thresh] = edge(I,'zerocross',…) 返回阈值。

9.2.2　各种边缘检测算子的效果比较

分别采用 Roberts 算子、Sobel 算子、Prewitt 算子、LOG 算子和 Canny 算子和零交叉方法检测图像“rice.tif”的边缘。

例程 9-5 对各种边缘检测算子的效果进行比较。

例程 9-5

```
I = imread('rice.tif');
BW1 = edge(I,'sobel');
BW2 = edge(I,'roberts');
BW3 = edge(I,'prewitt');
BW4 = edge(I,'log');
BW5 = edge(I,'canny');
h=fspecial('gaussian',5);
BW6 = edge(I,'zerocross',[],h);
subplot(2,3,1),imshow(BW1)
subplot(2,3,2),imshow(BW2)
subplot(2,3,3),imshow(BW3)
subplot(2,3,4),imshow(BW4)
subplot(2,3,5),imshow(BW5)
subplot(2,3,6),imshow(BW6)
```

结果如图 9-8 所示。

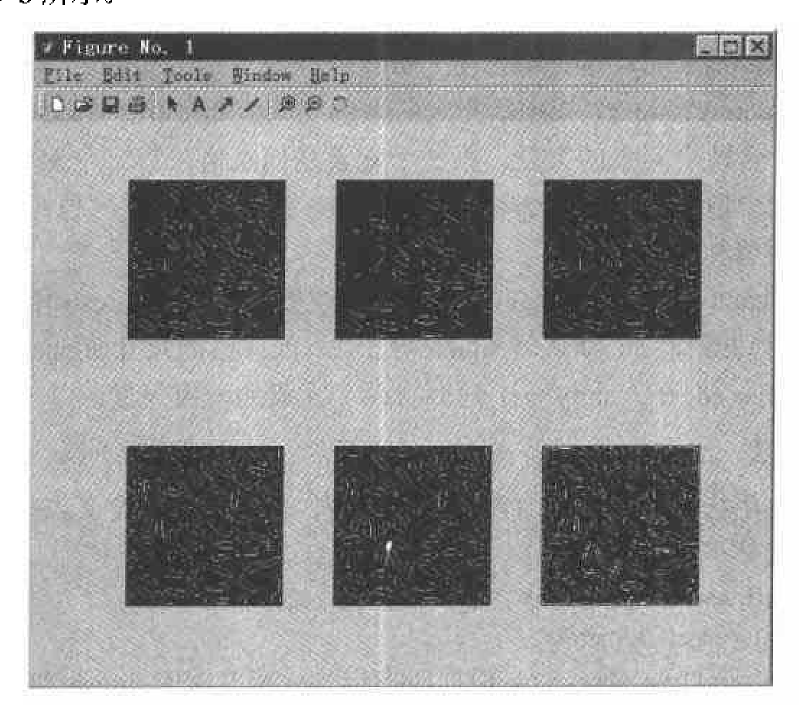


图 9-8　使用各种边缘检测算子得到的边缘图像

第 10 章　二值图像操作

二值图像中所有的像素只能从 0 和 1 这两个值中取，因此在 MATLAB 中，二值图像用一个由 0 和 1 组成的二维矩阵表示。这两个可取的值分别对应于关闭和打开，关闭表征该像素处于背景，而打开表征该像素处于前景。以这种方式来操作图像可以更容易识别出图像的结构特征。例如，在二值图像中我们可以非常容易地从图像背景中识别 MATLAB 对象。

本章主要内容：

- 二值形态学基本运算
- 二值形态学图像处理综合应用

10.1　引　　言

二值图像操作只返回与二值图像的形式或结构有关的信息，如果希望对其他类型的图像进行同样的操作，则首先要将其转换为二进制的图像格式，可以通过调用 MATLAB 提供的函数 im2bw 来实现，方法如下：

```
I=imread('cameraman.tif');
figure,imshow(I)
J=im2bw(I);
figure,imshow(J)
```

原始图像如图 10-1 所示。



图 10-1　cameraman.tif 原始图像

二值化处理后的图像如图 10-2 所示。



图 10-2　二值化的结果

从以上两幅图的比较可以看出，二值化的图像中只有纯黑和纯白两种灰度，因而原图像中的很多细节都丢失了。

10.2　二值形态学基本运算

二值形态学中的运算对象是集合，也就是二值矩阵，但实际上当涉及两个二值矩阵时并不把它们看做是对等的，通常设 ***A*** 为图像矩阵，***B*** 为结构元素矩阵，数学形态学运算是用 ***B*** 对 ***A*** 进行操作。实际上，结构元素本身也是一个图像矩阵。我们对每个结构元素矩阵指定一个原点，在 MATLAB 中称做中心像素，它是结构元素参与形态学运算的参考点，通常表示用户期望的像素，在 MATLAB 中，中心像素的定义如下：

```
floor ((size (SE) +1) /2)
```

其中，***SE*** 是结构要素矩阵。例如：

$$\boldsymbol{SE}=\begin{bmatrix}0&0&0&0\\0&1&1&1\\0&1&1&1\\0&1&1&1\end{bmatrix}$$

代码及结果如下：

```
floor((size(SE)+1)/2)
ans =2      2
```

10.2.1 膨胀

膨胀的算符为⊕，A 用 B 来膨胀写做 $A\oplus B$，这里先将 A 和 B 看做是所有取值为 1 的像素点的集合。其定义为：

$$A\oplus B=\left\{x\middle|\left[(\hat{B})_x\cap A\right]\neq\Phi\right\} \tag{10-1}$$

其中 $\hat{B}$ 表示 B 的映像，定义为：

$$\hat{B}=\left\{x\middle|x=-b,b\in B\right\} \tag{10-2}$$

$(\hat{B})_x$ 表示对 B 的映像进行位移 x，定义为：

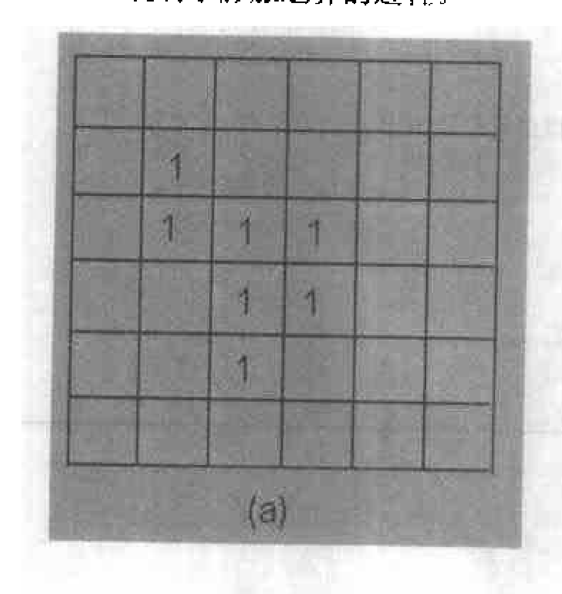
$$(M)_x=\left\{y\middle|y=a+x,a\in M\right\} \tag{10-3}$$

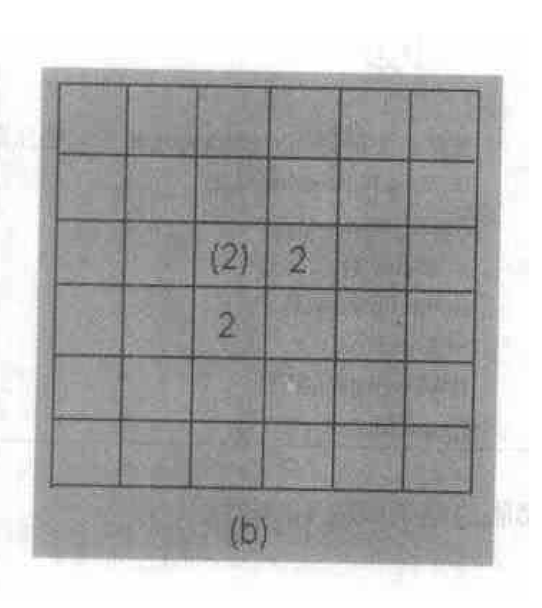
上式表明用 B 膨胀 A 的过程是，先对 B 做关于中心像素的映射，再将其映像平移 x，这里 A 与 B 映像的交集不为空集。换句话说，用 B 来膨胀 A 得到的集合是 $\hat{B}$ 的位移与 A 至少有一个非零元素相交时 B 的中心像素的位置的集合。根据这个解释，式（10-1）也可以写成：

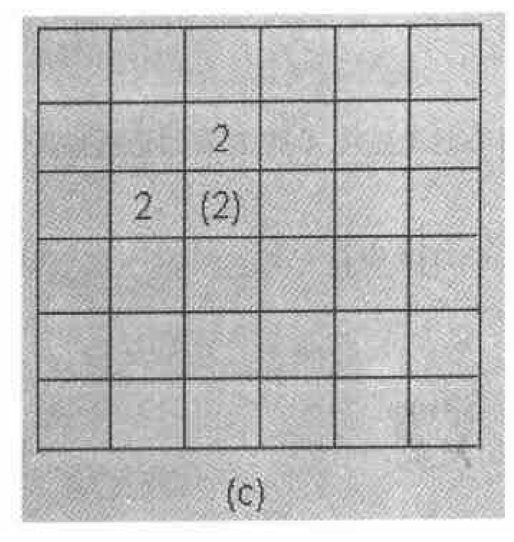
$$A\oplus B=\left\{x\middle|\left[(\hat{B})_x\cap A\right]\subseteq A\right\} \tag{10-4}$$

上式可帮助我们借助卷积的概念来理解膨胀操作。如果将 B 看做是一个卷积模板，膨胀就是先对 B 做关于中心像素的映射，再将映像连续地在 A 上移动而实现的。

图 10-3 说明了膨胀运算的过程。







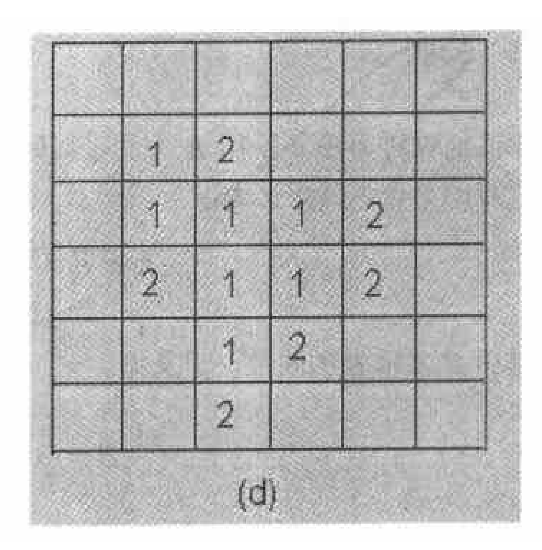
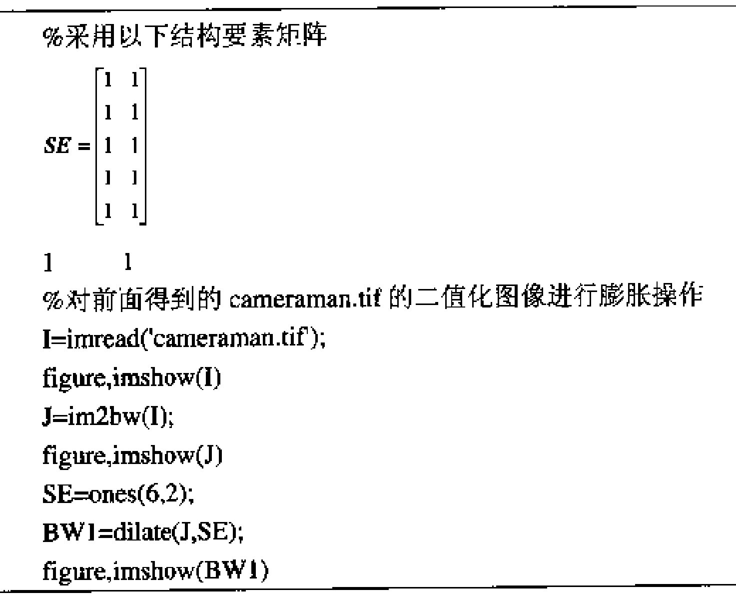


图 10-3　膨胀运算过程示例

在图 10-3 中，图（a）中填入“1”的部分为集合 A。图（b）中填入“2”的部分为集合 B，标有“(2)”的位置表示中心像素，这并不代表其所在位置处的像素的灰度值为 2，而只是为了与集合 A 相区分。图（c）是图（b）的映像。图（d）中标有“1”的像素为 A 中原来的位置，标有“2”的像素表示膨胀出来的部分，合起来就是膨胀的结果，也就是 $A \oplus B$。从图中很明显可以看到膨胀运算将原图像区域扩大了。

例程 10-1 用 dilate 函数来实现膨胀操作。

例程 10-1



%采用以下结构要素矩阵

$$SE = \begin{bmatrix} 1 & 1 \\ 1 & 1 \\ 1 & 1 \\ 1 & 1 \\ 1 & 1 \end{bmatrix}$$

```
1        1
%对前面得到的 cameraman.tif 的二值化图像进行膨胀操作
I=imread('cameraman.tif');
figure,imshow(I)
J=im2bw(I);
figure,imshow(J)
SE=ones(6,2);
BW1=dilate(J,SE);
figure,imshow(BW1)
```

膨胀的结果如图 10-4 所示。



图 10-4 使用 dilate 函数膨胀的结果

dilate 函数有多种形式：

```
BW2 = dilate(BW1,SE)
BW2 = dilate(BW1,SE,alg)
BW2 = dilate(BW1,SE,…,n)
```

BW2 = dilate(BW1,SE)使用二值结构要素矩阵 SE 对图像数据矩阵 BW1 执行膨胀操作。输入图像 BW1 的类型为 double 或 uint8，输出图像 BW2 的类型为 uint8。

BW2 = dilate(BW1,SE,alg)使用指定的算法执行膨胀操作，alg 是用来指定算法的字符串，可以取以下值：'spatial'（在空域内处理图像，这是默认值），'frequency'（在频域内处理图像）。这两种算法产生的结果是一样的，所不同的是处理速度和内存占用的多少。对于很大的图像和结构要素矩阵，频域算法比空域算法要快，但占用的内存要大得多。

BW2 = dilate(BW1,SE,…,n) 执行膨胀操作 n 次。



除非你的系统内存很大，否则不要选择频域算法。如果你的内存不足却选用了频域算法，则实际上比空域算法还要慢，因为要占用虚拟内存页。如果频域算法极度地降低了你的系统速度，或者产生了“内存不足”的消息，那么还是改用空域算法吧。

此外 MATLAB 提供了预定义的形态运算函数 bwmorph，其语法格式如下：

```
BW2 = bwmorph(BW1,operation)
BW2 = bwmorph(BW1,operation,n)
```

BW2 = bwmorph(BW1,operation) 对图像 BW1 做指定的形态学运算 operation。

BW2 = bwmorph(BW1,operation,n) 对图像 BW1 做 n 次指定的形态学运算 operation。N 可以是 Inf，在这种情况下，操作将持续到图像不再变化为止。

其中 BW1 可以是 double 型或 uint8 型，BW2 是 uint8 型，operation 可选以下值：

- 'bothat'，闭包运算，即先腐蚀，再膨胀，然后减去原图。

- 'erode'，用结构元素***ones***(3)腐蚀运算。
- 'shrink'，n = Inf ，将对象收缩成点。它消除像素，使没有孔的对象缩成一点，使有孔对象缩成外层边缘，在每个孔之间缩成一个相连的环。此项保持欧拉数。
- 'bridge'，联接运算，将原来不相连的像素连接起来，如

$$\begin{bmatrix}1 & 0 & 0\\1 & 0 & 1\\0 & 0 & 1\end{bmatrix} 变成 \begin{bmatrix}1 & 0 & 0\\1 & 1 & 1\\0 & 0 & 1\end{bmatrix}$$

- 'fill'，填充孤立的内部像素点（中间是0，周围是1），类似于以下情况：

$$\begin{bmatrix}1 & 1 & 1\\2 & 0 & 1\\1 & 1 & 1\end{bmatrix}$$

- 'skel'，n = Inf ，消除对象边缘上的点，但不使对象分裂。剩下的像素构成图像的骨架。此项保持欧拉数。
- 'clean'，清除孤立的亮点（中间是1，而周围都是0），类似于以下的情况：

$$\begin{bmatrix}0 & 1 & 0\\0 & 0 & 0\\0 & 1 & 0\end{bmatrix}$$

- 'hbreak'，断开H型连接的像素，例如

$$\begin{bmatrix}1 & 1 & 1\\0 & 1 & 0\\1 & 1 & 1\end{bmatrix} 变成 \begin{bmatrix}1 & 1 & 1\\0 & 0 & 0\\1 & 1 & 1\end{bmatrix}$$

- 'spur'，去掉小短枝像素，例如

$$\begin{bmatrix}0 & 0 & 0 & 0\\0 & 0 & 0 & 0\\0 & 0 & 1 & 0\\0 & 1 & 0 & 0\\1 & 1 & 0 & 0\end{bmatrix} 变成 \begin{bmatrix}0 & 0 & 0 & 0\\0 & 0 & 0 & 0\\0 & 0 & 0 & 0\\0 & 1 & 0 & 0\\1 & 1 & 0 & 0\end{bmatrix}$$

- 'close'，执行二值闭运算（先膨胀，再扩张）。
- 'majority'，若像素的8个邻域中有5个以上的像素为1，则像素值为1，否则为0。
- 'thicken'，n = Inf，在对象的外部不断增加像素，但要保证所增加的像素不会导致原来不连接的对象成为8-连接，此项保持欧拉数。
- 'diag'，使用对角线填充来消除背景的8-连接。例如

$$\begin{bmatrix}0&1&0\\1&0&0\\0&0&0\end{bmatrix}\text{变成}\begin{bmatrix}0&1&0\\1&1&0\\0&0&0\end{bmatrix}$$

- 'open'，开启运算（先腐蚀，然后膨胀）。
- 'thin'，n = Inf，将对象细化成线。如果是无孔的对象，则缩成最小连接的一笔；如果是有孔的对象，则缩成外层边缘，而在每个孔之间缩成相连的环。此项保持欧拉数。
- 'dilate'，用结构元素 ***ones***(3)膨胀。
- 'remove'，去掉内部像素点，即若像素的四邻域都为 1，则像素值为 0，就使得边缘的像素是亮的。
- 'tophat'，用原图减去开启运算后的图。

以上这些不同的功能将在以后分别介绍。

例程 10-2 利用 bwmorph 函数来实现膨胀。

例程 10-2

```
I=imread('cameraman.tif');
figure,imshow(I)
J=im2bw(I);
figure,imshow(J)
BW1= bwmorph(J,'dilate');
figure,imshow(BW1)
```

输出结果如图 10-5。



图 10-5　使用 bwmorph 函数膨胀的结果

虽然利用 BW1= bwmorph(J,'erode')函数时，腐蚀的结构要素矩阵是 ***ones***(3)：

$$ones = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

但与前面利用 dilate 函数膨胀的结果相比，差别不大，原因在于所采用的结构要素矩阵相对于整个图像矩阵来说很小，因而这两种情况下作用的结果很相似。

10.2.2 腐蚀

腐蚀的算符为Θ，A 用 B 来腐蚀写做 $A\Theta B$。其定义为：

$$A\Theta B = \{x | (B)_x \subseteq A\} \tag{10-5}$$

上式表明 A 用 B 来腐蚀的结果是所有 x 的集合，其中 B 平移 x 后仍在 A 中。换句话说，用 B 来腐蚀 A 得到的集合是 B 完全包括在 A 中时 B 的原点位置的集合。

图 10-6 举例说明了腐蚀运算的过程。图（a）中填入“1”的部分为集合 A。图（b）中填入“2”的部分为集合 B，标有“(2)”的位置表示中心像素，这并不代表其所在位置处的像素的灰度值为 2，而只是为了与集合 A 相区分。图（c）中标有“1”的像素表示腐蚀后 A 中保留下来的像素，标有“2”的像素表示 A 中被腐蚀掉的像素。从图中很明显可以看到腐蚀运算将原图像区域缩小了。

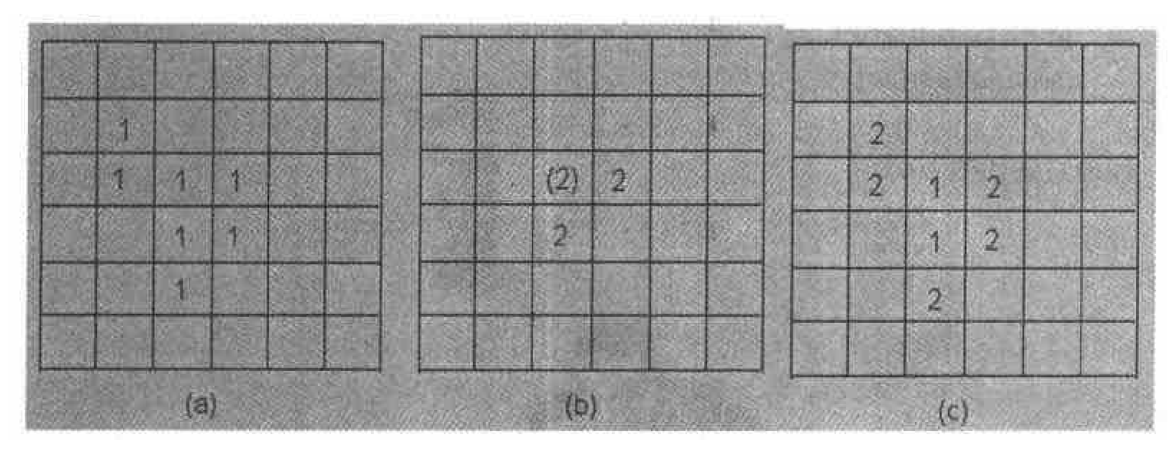


图 10-6　腐蚀运算过程示例



腐蚀公式中的运算符Θ与其他书中有出入，请读者注意。

例程 10-3 用 erode 函数来实现腐蚀操作。

例程 10-3

```
%采用以下结构要素矩阵
```

$$SE=\begin{bmatrix}1&0&0&0&0\\0&1&0&0&0\\0&0&1&0&0\\0&0&0&1&0\\0&0&0&0&1\end{bmatrix}$$

```
%对前面得到的 cameraman.tif 的二值化图像进行腐蚀操作
I=imread('cameraman.tif');
figure,imshow(I)
J=im2bw(I);
figure,imshow(J)
SE=eye(5);
BW1=erode(J,SE);
figure,imshow(BW1)
```

腐蚀的结果如图 10-7 所示。



图 10-7　使用 erode 函数腐蚀的结果

erode 函数有几种形式：

```
BW2 = erode(BW1,SE)
BW2 = erode(BW1,SE,alg)
BW2 = erode(BW1,SE,…,n)
```

BW2 = erode(BW1,SE) 使用二值结构要素矩阵 SE 对图像数据矩阵 BW1 执行腐蚀操作。输入图像 BW1 的类型为 double 或 uint8，输出图像 BW2 的类型为 uint8。

BW2 = erode(BW1,SE,alg) 使用指定的算法执行腐蚀操作，alg 是用来指定算法的字符串，可以取以下值：'spatial'（在空域内处理图像，这是默认值），'frequency'（在频域内处理图像）。这两种算法产生的结果是一样的，所不同的是处理速度和内存占用的多少。对于很大的图像和结构要素矩阵，频域算法比空域算法要快，但占用的内存要大得多。

BW2 = erode(BW1,SE,…,n) 执行腐蚀操作 n 次。



除非你的系统内存很大，否则不要选择频域算法；如果你的内存不足却选用了频域算法，则实际上比空域算法还要慢，因为要占用虚拟内存页。如果频域算法极度的降低了你的系统速度，或者产生了“内存不足”的消息，那么还是改用空域算法吧。

例程 10-4 利用 bwmorph 函数来实现腐蚀。

例程 10-4

```
I=imread('cameraman.tif');
figure,imshow(I)
J=im2bw(I);
figure,imshow(J)
BW1= bwmorph(J,'erode');
figure,imshow(BW1)
```

输出效果如图 10-8 所示。



图 10-8　使用 bwmorph 函数腐蚀的结果

虽然利用 BW1= bwmorph(J,'erode')函数时，腐蚀的模板是 ***ones***（3）：

$$\boldsymbol{ones}=\begin{bmatrix}1 & 1 & 1\\1 & 1 & 1\\1 & 1 & 1\end{bmatrix}$$

但与前面利用 erode 函数腐蚀的结果相比，只有细小的差别，原因在于所采用的结构要素矩阵相对于整个图像矩阵来说很小，因而在两种情况下作用的结果很相似。

10.2.3　膨胀与腐蚀的对偶性

膨胀与腐蚀这两种运算是紧密联系在一起的，一种运算对目标的操作相当于另一种运算对图像背景的操作。膨胀与腐蚀的对偶性可表示为：

$$(A \oplus B)^c = A^c \Theta \hat{B} \tag{10-6}$$

$$(A \Theta B)^c = A^c \oplus B \tag{10-7}$$

图 10-9 说明了膨胀与腐蚀的对偶性。图（a）中填入“1”的部分为集合 A。图（b）中填入“2”的部分为集合 B，标有“(2)”的位置表示中心像素，这并不代表其所在位置处的像素的灰度值为 2，而只是为了与集合 A 相区分。图（c）是腐蚀的结果，标有“1”的像素表示腐蚀后 A 中保留下来的像素，标有“2”的像素表示 A 中被腐蚀掉的像素。图（d）中标有“1”的像素为 A 中原来的位置，标有“2”的像素表示膨胀出来的部分，合起来就是膨胀的结果。图（e）中值为“1”的所有像素构成了 A 的补集。图（f）是 B 的映像。图（g）是 $A^c \Theta \hat{B}$ 的结果，也就是图（f）腐蚀图（e）的结果，其中值为“1”的所有像素是原来图（e）中未被腐蚀掉的像素，而值为“2”的像素是原来图（e）中被腐蚀掉的像素。图（h）是 $A^c \oplus \hat{B}$ 的结果，也就是由图（f）膨胀图（e）的结果，其中值为“1”的所有像素正是原来图（e）中的像素，而值为“2”的像素是膨胀出来的。比较图（d）和图（g）可以验证式（10-6），比较图（c）和图（h）可以验证式（10-7）。

	1				
	1	1	1		
		1	1		
		1			

(a)

		(2)	2		
		2			

(b)

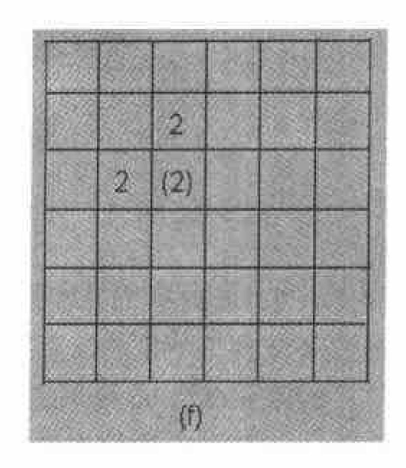
	2				
	2	1	2		
		1	2		
		2			

(c)

	1	2			
	1	1	1	2	
	2	1	1	2	
		1	2		
		2			

(d)

(e)



(g)

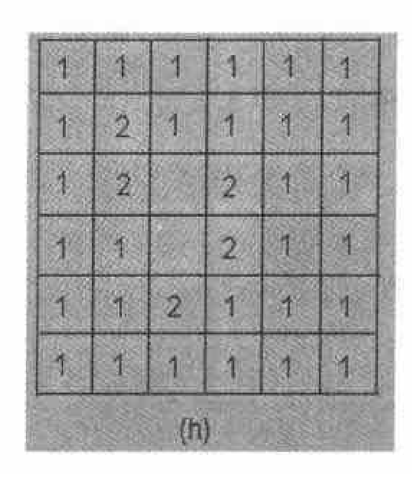


图 10-9　膨胀与腐蚀的对偶性图解

10.2.4　开启和闭合

由于膨胀和腐蚀并不是互为逆运算，所以可以将它们级联结合使用。开启就是先对图像进行腐蚀，然后膨胀其结果。闭合就是先对图像进行膨胀，然后腐蚀其结果。它们也是数学形态学中的重要运算。

开启的运算符为$\circ$，A 用 B 来开启写做 $A \circ B$，其定义为：

$$A \circ B = (A \Theta B) \oplus B \tag{10-8}$$

闭合的运算符为$\bullet$，A 用 B 来闭合写做 $A \bullet B$，其定义为：

$$A \bullet B = (A \oplus B) \Theta B \tag{10-9}$$

开启和闭合这两种运算都可以除去比结构元素小的特定图像细节，同时保证不产生全局几何失真。开启运算可以把比结构元素小的突刺滤掉，切断细长搭接而起到分离作用。闭合运算可以把比结构元素小的缺口或孔填充上，搭接短的间断而起到连通作用。

开启和闭合可以从图像中提取与其结构元素相匹配的形状，由以下开启特性定理和闭合特性定理可以得到：

$$A \circ B = \{x \in A | 对某些 t \in A \Theta B, x \in (B)_t 和 (B)_t \subseteq A\} \qquad (10\text{-}10)$$

$$A \bullet B\{x | x \in (\hat{B})_t \Rightarrow (\hat{B})_t \cap A \neq \Phi\} \qquad (10\text{-}11)$$

式（10-10）表明，用 B 开启 A 就是选出了 A 中的某些与 B 相匹配的点，这些点可由完全包含在 A 中的结构元素 B 的平移得到。式（10-11）表明，用 B 对闭合 A 的结果包括所有满足如下条件的点，即该点可被映射和位移的结构元素覆盖时，A 与经过映射和位移的结构元素的交集不为零。

例程 10-5 以 circbw.tif 图为例展示开启的效果，采用一个 40×30 的全 1 的结构要素矩阵，闭合操作的实现代码如下：

例程 10-5

```
I=imread('circbw.tif');
figure,imshow(I)
SE=ones(40,30);
BW1=erode(I,SE);
figure,imshow(BW1)
BW2=dilate(BW1,SE);
figure,imshow(BW2)
```

输出结果如图 10-10 至图 10-13 所示。

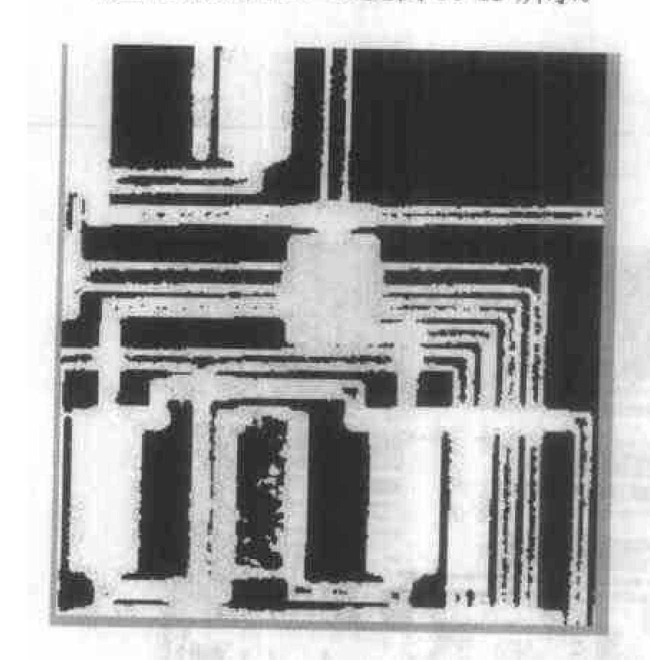


图 10-10　circbw.tif 原始图像

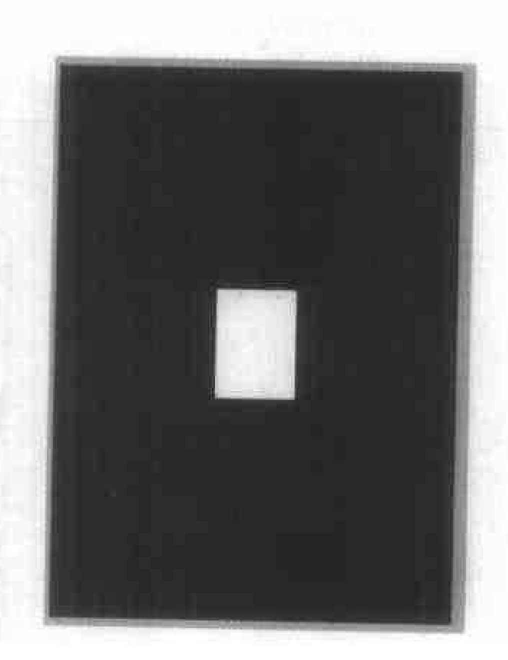


图 10-11　结构要素矩阵

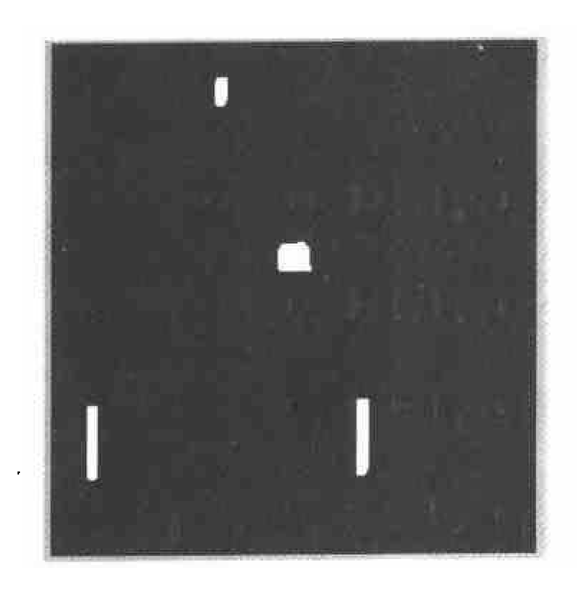


图 10-12　开启的腐蚀结果

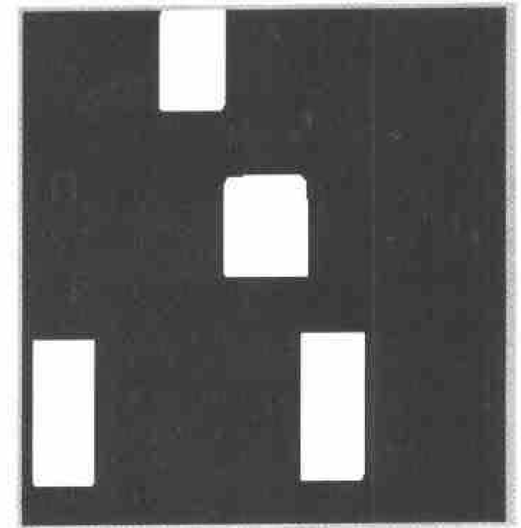


图 10-13　开启的最终结果

从以上 4 幅图可以看出，原始图像中有很多比结构要素矩阵小的细小轮廓，开启操作将原始图像中的细节去掉了，而将其中较大的方形轮廓按照原来的大小显示出来了。

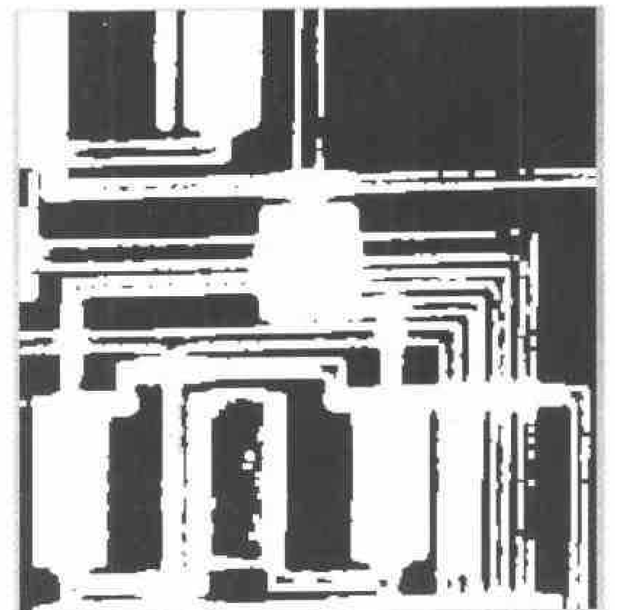
例程 10-6 利用 bwmorph 函数来实现开启。

例程 10-6

```
I=imread('circbw.tif');
figure,imshow(I)
BW1= bwmorph(I,'open');
figure,imshow(BW1)
```

效果如图 10-14、图 10-15 所示。

图 10-14　*ones*(3)结构要素矩阵



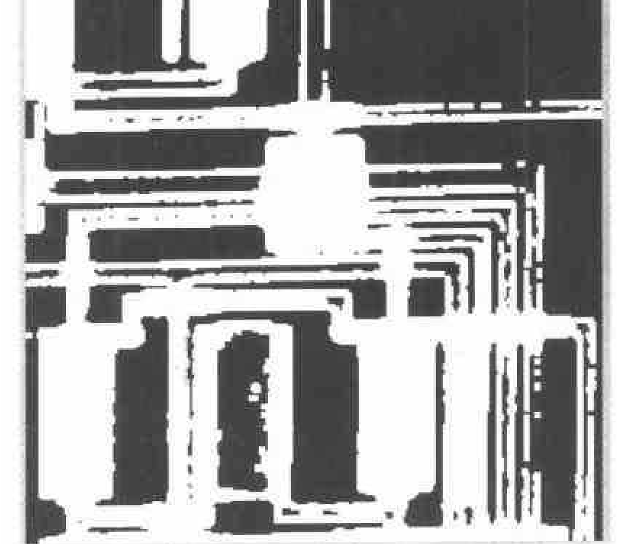


图 10-15　利用 bwmorph 函数开启的结果

与前面利用 erode 函数和 dilate 函数的结果相比差别很大，造成这种结果的主要原因在结构要素矩阵。调用 BW1= bwmorph(I,'open')函数时的结构要素矩阵如图 10-14 所示，同图 10-11 的结构要素矩阵相比要小很多，而在 circbw.tif 原始图像中比图 10-14 中的结构要素小的图像细节很少，所以操作的结果就是只有少数细小的连接处断开了，而不像图 10-13 那样效果明显。所以如果想要达到特定的目的，最好采用原始的方法实现开启操作，也就是先腐蚀，然后膨胀，采用这种方法时结构要素矩阵是可由用户根据需要随意设定的，会比较理想。

闭合操作也是同样的道理，采用原始的方法，也就是先膨胀，然后腐蚀，这种方法可由用户控制结构要素，根据需要进行调节，所以比调用 BW1= bwmorph(I,'close')函数时处理的效果要更加理想。

10.3　二值形态学图像处理综合应用

本节将讲述二值形态学图像处理的几个综合性的例子，这些应用涉及噪声滤除、边界提取、对象标注图像的特性度量、细化与骨架提取、查找表操作、区域填充及对象提取等。

10.3.1　噪声滤除

将开启和闭合结合起来可构成形态学噪声滤除器。下面将 saturn.tif 图像二值化，如图 10-16 所示。将二值化得到的图像加入椒盐噪声，这种噪声在亮的图像区域内是暗点，而在暗的图像区域中是亮点，如图 10-17 所示。首先对有噪声图像进行开启操作，由于这里的结构要素矩阵比噪声的尺寸要大，所以开启的结果是将背景上的噪声点去除了，如图 10-18 所示。最后对前一步得到的图像进行闭合操作，将土星上的噪声去掉了，如图 10-19 所示。

例程 10-7 将开启和闭合结合起来构成形态学噪声滤除器。

例程 10-7

```
I1=imread('saturn.tif');
I2=im2bw(I1);
figure,imshow(I2)
I3=imnoise(I2,'salt & pepper');
figure,imshow(I3)
I4= bwmorph(I3,'open');
figure,imshow(I4)
I5= bwmorph(I4,'close');
figure,imshow(I5)
```

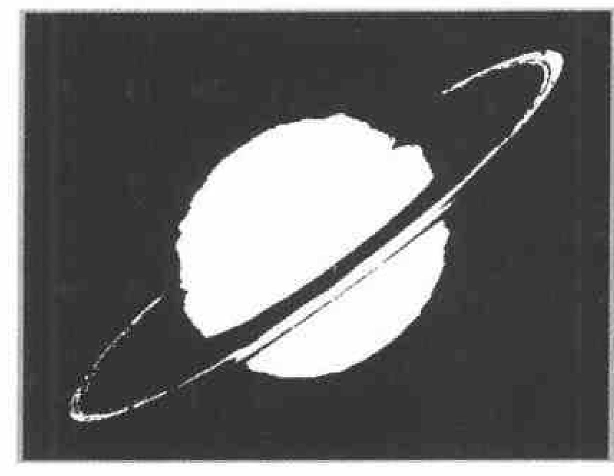


图 10-16 将 saturn.tif 二值化的图像

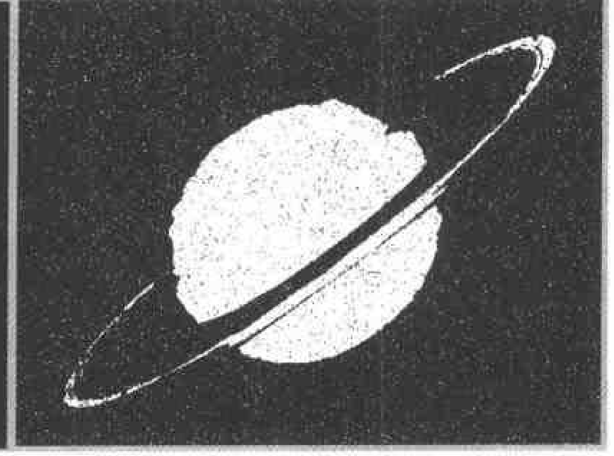


图 10-17 加入椒盐噪声的图像

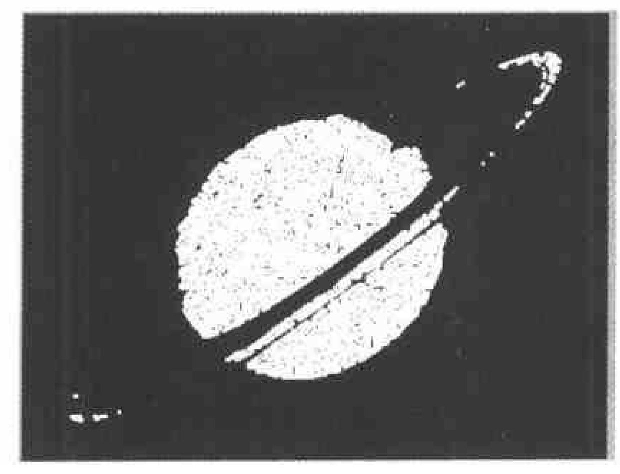


图 10-18 开启操作所得的图像

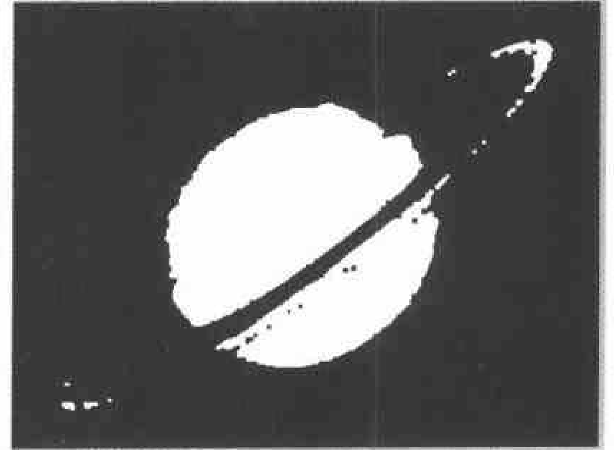


图 10-19 再进行闭合操作所得的图像

虽然采用以上方法达到了去除噪声的目的，但是比较图 10-18 和图 10-16 就会发现，处理后的图像在光环上有像素点损失，得到的结果和原图像不完全一样。通过分析可知出现这样的结果是由于光环上有一些不连续的像素点的尺寸比结构要素矩阵的尺寸要小，因此在除噪的过程中也当做噪声一样滤除了。

根据此方法的特点可以知道，此方法适用的图像类型是图像中的对象尺寸都比较大，没有细小的细节，对这种类型的图像除噪的效果会比较好。

10.3.2 边界提取

1. 二值图像中的对象

在二值图像中，所谓的对象就是值为 1 的连接在一起的像素的集合。例如，以下所示的矩阵就代表了包含一个简单的3×3方形对象的二值图像。矩阵中值为 0 的像素表示图像的背景。

0　0　0　0　0　0
0　1　1　1　0　0

0	1	1	1	0	0
0	1	1	1	0	0
0	0	0	0	0	0
0	0	0	0	0	0

2．4-连接和 8-连接边沿

对于大多数操作来说，二值图像中对象的识别依赖于确定图像中相邻像素是否连接的约定方式。MATLAB 通常使用两种连接方式：4-连接和 8-连接。

在 4-连接边沿约定方式中，所有与用户期望像素点接触的 8 个像素点中，只有垂直方向的 4 个像素均为可能连接的像素，如图 10-20 所示。

在 8-连接边沿约定方式中，所有与用户期望像素点接触的 8 个像素点均为可能连接的像素，如图 10-21 所示。

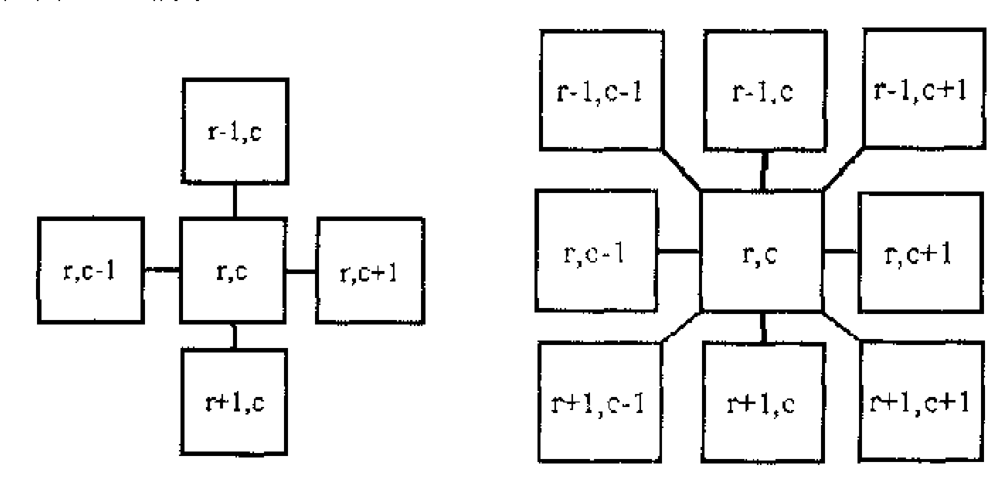


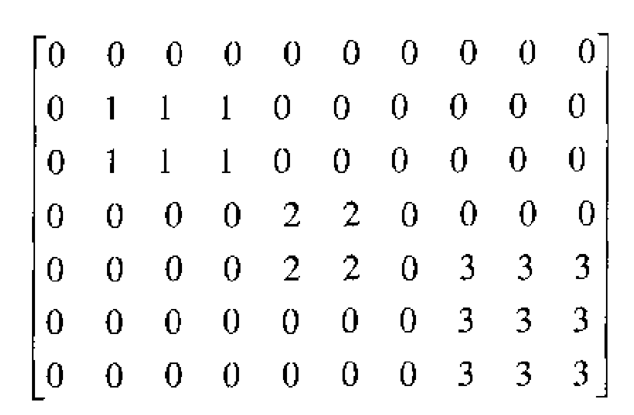
图 10-20　4-连接边沿示例　　　　图 10-21　8-连接边沿示例

根据以上定义，下面的示例矩阵所表示的二值图像在 4-连接边沿约定下包含三个对象，而在 8-连接边沿约定下包含两个对象。

示例矩阵：

$$
\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1
\end{bmatrix}
$$

4-连接边沿约定下三个对象分别如下：



$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 2 & 0 & 3 & 3 & 3 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 3 & 3 & 3 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 3 & 3 & 3 \end{bmatrix}$$

8-连接边沿约定下三个对象分别如下：

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 2 & 2 & 2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 2 & 2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 2 & 2 \end{bmatrix}$$

在 MATLAB 图像处理工具箱中提供了函数 bwperim 来提取二值图像中对象的边界像素。语法格式如下：

```
BW2 = bwperim(BW1,n)
```



确定图像 BW1 中某像素为边界像素的标准是该像素的值为 1，其邻域中至少有一个像素是非零的。用户可以用 4-连接边沿约定或 8-连接边沿约定来确定对象边界。对应着 *n* 可取的值有 4 和 8，默认时采用 4-连接边沿约定。

例程 10-8 以 blood1.tif 图像的二值化图像为例进行边界识别。

例程 10-8

```
I1=imread('blood1.tif');
I2=im2bw(I1);
figure,imshow(I2)
I4= bwperim(I2);
figure,imshow(I4)
```

结果如图 10-22 和图 10-23 所示。

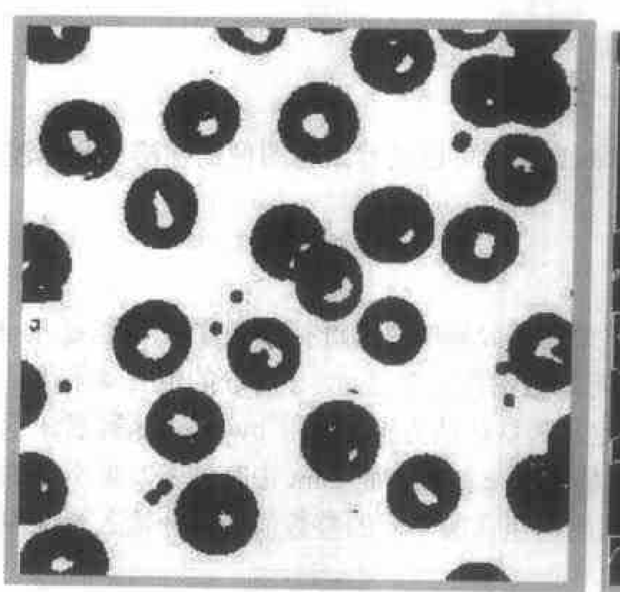


图 10-22 blood1.tif 图像的二值化图像

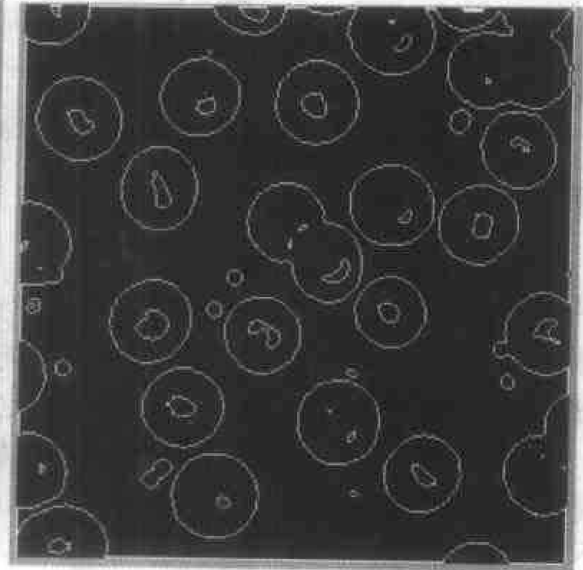


图 10-23 bwperim 函数处理的结果

另外，利用 bwmorph 函数可以得到同样的边界识别效果，代码如下：

```
I1=imread('blood1.tif');
I2=im2bw(I1);
figure,imshow(I2)
I4= bwmorph(I2,'remove');
figure,imshow(I4)
```

处理结果如图 10-24 所示。

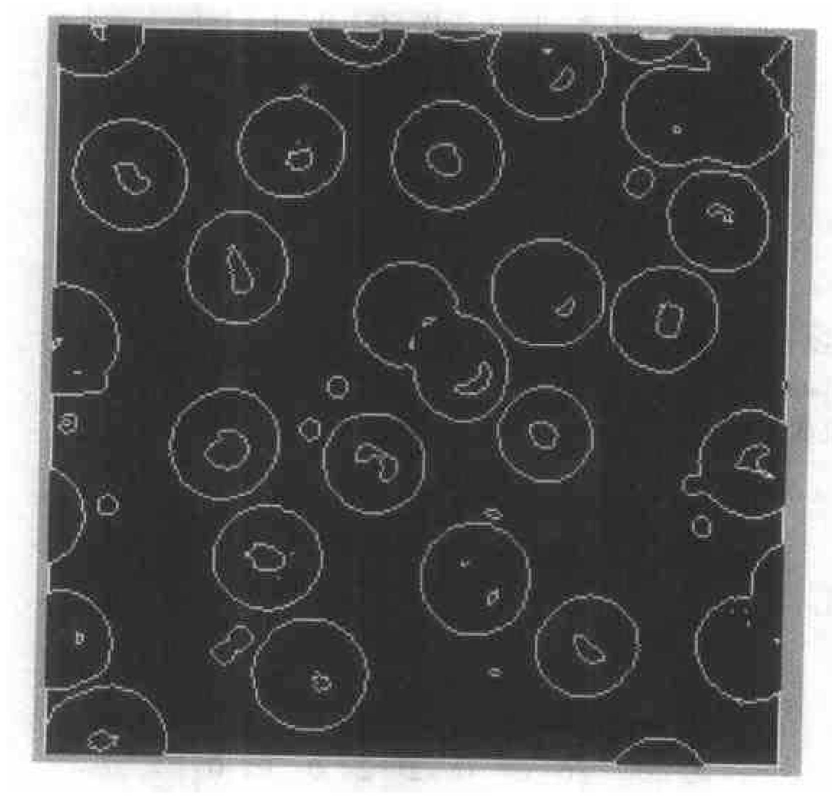


图 10-24 bwmorph 函数处理的结果

10.3.3 对象标注

在 MATLAB 图像处理工具箱中，bwlabel 函数可以进行二值图像的连接部分标注操作。语法是：

```
L= bwlabel(BW,n)
[L, num]= bwlabel(BW,n)
```

这种形态操作主要用于对二值图像 BW 中各个分离的对象进行标识，BW 可以是 double 型或 uint8 型。用户指定输入图像 BW 和特定的边沿约定类型 n，n 可取的值有 4 和 8，分别对应着 4-连接边沿约定和 8-连接边沿约定，其默认值为 8。调用 bwlabel 函数后，返回一个与输入图像相同大小的数据矩阵 L，L 是 double 型的，而 num 中存储着对象个数的数据。有了这个输出数据矩阵，我们就可以利用它包含的不同的整数值来区分输入图像中的不同对象。

例如，假设有二进制图像如下：

```
I(8,8)=0;
I(2:4,2:3)=1;
I(5:7,4:5)=1;
I(2,6)=1;
I(2:3,7:8)=1
```

$$I=\begin{bmatrix}0&0&0&0&0&0&0&0\\0&1&1&0&0&1&1&1\\0&1&1&0&0&0&1&1\\0&1&1&0&0&0&0&0\\0&0&0&1&1&0&0&0\\0&0&0&1&1&0&0&0\\0&0&0&1&1&0&0&0\\0&0&0&0&0&0&0&0\end{bmatrix}$$

调用 bwlabel 函数，指定 4-连接边沿约定：

```
[L,num]=bwlabel(I,4)
```

得到的结果如下：

$$L=\begin{bmatrix}0&0&0&0&0&0&0&0\\0&1&1&0&0&3&3&3\\0&1&1&0&0&0&3&3\\0&1&1&0&0&0&0&0\\0&0&0&2&2&0&0&0\\0&0&0&2&2&0&0&0\\0&0&0&2&2&0&0&0\\0&0&0&0&0&0&0&0\end{bmatrix}$$

num =3

调用 bwlabel 函数，指定 8-连接边沿约定：

```
[L,num]=bwlabel(I,8)或[L,num]=bwlabel(I)
```

得到的结果如下：

$$L = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 2 & 2 & 2 \\ 0 & 1 & 1 & 0 & 0 & 0 & 2 & 2 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

num =2

我们还可以将 bwlabel 函数与 find 函数结合起来使用，可以得到所指定对象中各像素的矩阵矢量，例如，返回上例中，对象 2 中各像素的坐标：

```
[r,c] = find(bwlabel(I)==2)
rc=[r,c]
```

得到的结果如下：

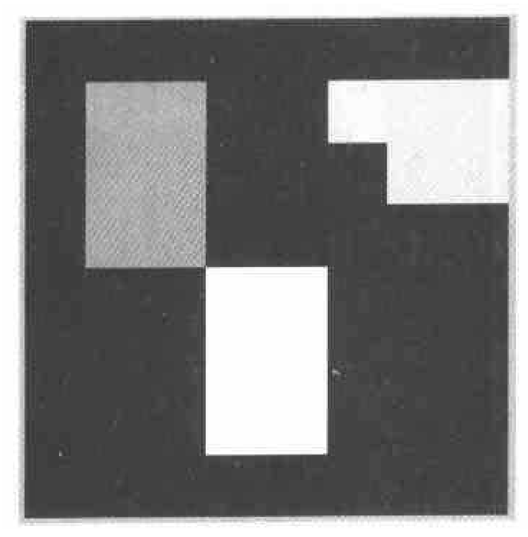
$$rc = \begin{bmatrix} 2 & 6 \\ 2 & 7 \\ 3 & 7 \\ 2 & 8 \\ 3 & 8 \end{bmatrix}$$

从例子中可以看出，输出矩阵不是一个二进制图像，访问该矩阵的一个有效方法是将其显示为伪彩色索引图像。将每个对象都显示成不同的颜色，所以在这样的显示图像中，对象更容易从整个图像中分辨出来，如下面的例子很好地说明了这一点。

例程 10-9 指定 4-连接边沿约定或 8-连接边沿约定。

例程 10-9

```
L=bwlabel(I,4);
map=[0 0 0;jet(3)];
imshow(L+1,map,'notruesize')
%显示的图像如图 10-25 所示
%指定 8-连接边沿约定
L=bwlabel(I,8);
map=[0 0 0;jet(3)];
imshow(L+1,map,'notruesize')
%显示的图像如图 10-26 所示
```



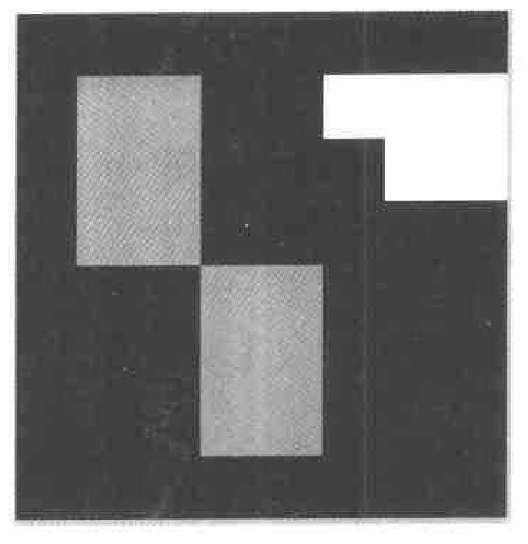




图 10-25　4-连接边沿约定下的对象　　图 10-26　8-连接边沿约定下的对象

10.3.4　图像的特性度量

当我们进行图像处理时，可能会希望获取图像中某些特性的变化信息。例如，在进行放大操作时，可能想知道到底有多少个像素值从 off 状态变成 on 状态，或者想知道图像中的对象数是否改变。为此，MATLAB 的图像处理工作提供了两个函数：bwarea 函数和 bweuler 函数。

1．图像的面积

bwarea 函数估计二值图像中对象的面积。这里所谓的面积是指图像前景的大小度量。粗略地讲，面积的大小约等于该图像中值为 on 的像素的数目，但并不是完全等于值为 on 的像素的数目，因为不同方式的像素所占的比重也不同。这个函数的语法很简单，格式如下：

```
total = bwarea(BW)
```

其中，BW 可以是 uint8 型或 double 型，total 是 double 型。

bwarea 函数估计二值图像中对象的面积，算法是：把图像中所有值为 on 的像素的面积相加。单个像素的面积是由其 2×2 邻域确定的，有 6 种不同的方式，分别表示了 6 种不同的面积：

- 有 0 个 on 像素的方式，area =0
- 有 1 个 on 像素的方式，area =1/4
- 有 2 个相邻的 on 像素的方式，area =1/2
- 有 2 个对角的 on 像素的方式，area =3/4
- 有 3 个 on 像素的方式，area =7/8
- 有 4 个 on 像素的方式，area =1



切记每个像素都是2×2邻域的 4 个不同位置之一，这意味着，当中间一个像素的值为 on，周围的像素都为 off 时，总的面积为 1。

下面以 circles.tif 图像为例，计算其中对象的面积，代码如下：

```
I= imread('circles.tif');
imshow(I);
area=bwarea(I)
[m,n]=size(I)
```

结果为：

```
area =
        15799
m =
        256
n =
        256
```

还可以通过 bwarea 函数来计算对图像进行操作前后图像中对象的面积的变化，例如对 circles.tif 图像进行腐蚀，代码如下：

```
SE=ones(5)
```

$$\boldsymbol{SE}=\begin{bmatrix}1&1&1&1&1\\1&1&1&1&1\\1&1&1&1&1\\1&1&1&1&1\\1&1&1&1&1\end{bmatrix}$$

```
I2=erode(I,SE);
figure,imshow(I2)
reduce=(bwarea(I)-bwarea(I2))/bwarea(I)
```

结果为：

```
reduce =
    0.1813
```

腐蚀前、后的图像如图 10-27 和图 10-28 所示。

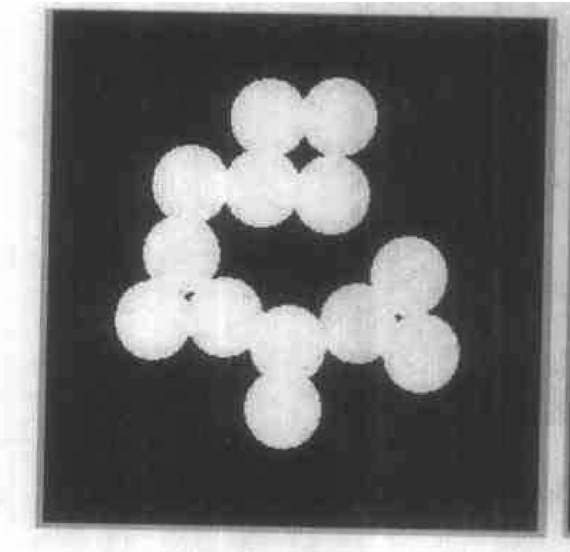


图 10-27 circles.tif 原始图像

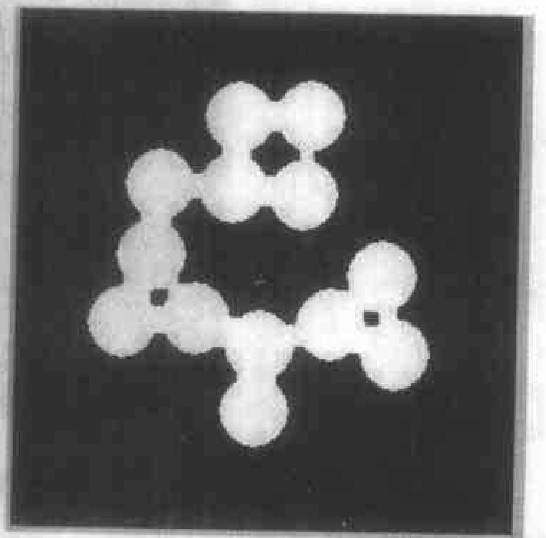


图 10-28 腐蚀后的 circles.tif 图像

另外，imfeature 函数也可以计算对象的面积，但与 bwarea 函数稍有不同，imfeature 函数得到的面积是对象中像素的实际个数。例如，对 circles.tif 图像调用 imfeature 函数，代码如下：

```
area=imfeature(I,'area')
输出：
area =
            area: 15743
```

两个函数计算所得的结果不同，反映了它们的差别。area=imfeature(I,'area')函数调用中，第二个参数指定对图像进行的测度类型，上例中选择了'area'，还可以有其他取值。imfeature 函数的语法如下：

```
stats = imfeature(L,measurements)
stats = imfeature(L,measurements,n)
```

它对标注矩阵 L 中的每个已标注的对象计算一组测度。L 可以通过 bwlabel 函数的计算得到，其中的正整数元素对应着图像中的不同对象，返回值 stats 是个结构数组，显示了由 measurements 指定的对每个对象的不同测度。n 可取的值有 4 和 8，分别对应着 4-连接边沿约定和 8-连接边沿约定，默认值为 8；当 measurements 取为'FilledImage'、'FilledArea'和'EulerNumber'时，才需要用到 n。 measurements 可以是由逗号隔开的字符串列表，可包括以下字符串：'Area'、'Centroid'、'BoundingBox'、'MajorAxisLength'、'MinorAxisLength'、'Eccentricity'、'Orientation'、'Image'、'FilledImage'、'FilledArea'、'ConvexHull'、'ConvexImage'、'ConvexArea'、'EulerNumber'、'Extrema'、'EquivDiameter'、'Solidity'、'Extent'、'PixelList'，或者也可以是'all'或'basic'。

下面以 text.tif 图像为例，运行以下代码：

```
BW = imread('text.tif');
L = bwlabel(BW);
stats = imfeature(L,'all');
stats(23)
```

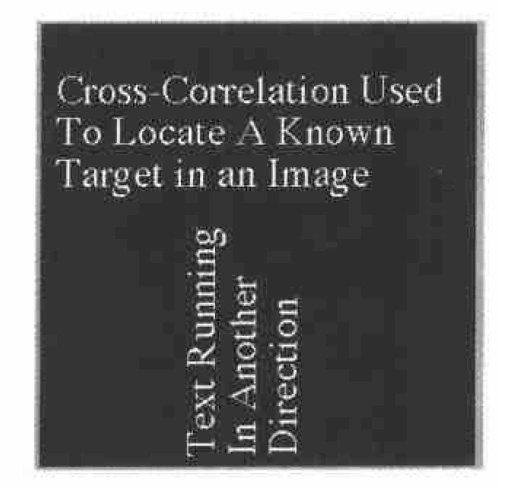


图 10-29　text.tif 图像

运行的结果如下：

```
ans =
```

```
Area: 89
Centroid: [95.6742 192.9775]
BoundingBox: [87.5000 184.5000 16 15]
MajorAxisLength: 19.9127
MinorAxisLength: 14.2953
Eccentricity: 0.6961
Orientation: 9.0845
ConvexHull: [28x2   double]
ConvexImage: [15x16 uint8 ]
ConvexArea: 205
Image: [15x16 uint8 ]
FilledImage: [15x16 uint8 ]
FilledArea: 122
EulerNumber: 0
Extrema: [ 8x2   double]
EquivDiameter: 10.6451
Solidity: 0.4341
Extent: 0.3708
PixelList: [89x2   double]
```

2. 欧拉数

在几何理论中，欧拉数是图像的一种拓扑度量。欧拉数等于图像中所有对象的总数减去这些对象中洞孔的数目。函数 bweuler 可以返回二值图像的欧拉数，语法格式如下：

```
eul = bweuler(BW,n)
```

其中，BW 是输入的二值图像，可以是 double 型或 uint8 型；n 可以取 4 或 8，分别对应着 4-连接边沿约定和 8-连接边沿约定，默认值为 8；返回值 eul 是 double 型的。

下面以图 10-27 中的 circles.tif 原始图像为例，运行代码如下：

```
I= imread('circles.tif');
imshow(I)
bweuler(I)
```

结果为：

```
ans =
        -2
```

此外，也可调用 imfeature 函数来计算欧拉数，代码如下：

```
number=imfeature(I,' EulerNumber ')
```

可以得到同样的结果：

```
number =
  EulerNumber: -2
```

10.3.5 细化与骨架提取

细化可用两步腐蚀来实现。第一步是正常的腐蚀，但它是有条件的，也就是说，那些被标为可除去的像素点并不立即消去。在第二步中，只将那些消除后并不破坏连通性的点

消除，否则保留。以上每一步都是一个 3×3 邻域运算。细化将一个曲线形物体细化为一条单像素宽的线，从而图形化地显示出其拓扑性质。

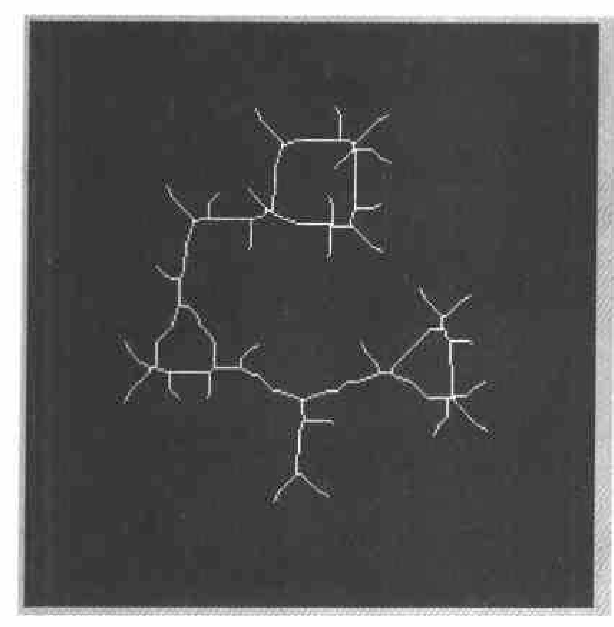
抽骨架是与细化有关的一种运算，也称为中轴变换或焚烧草地技术。中轴是所有与物体在两个或更多非邻接边界点处相切的圆心的轨迹。但抽骨架很少通过在物体内拟和圆来实现。在概念上，中轴可设想成按如下方式形成：想像一片与物体形状相同的草，沿其外围各点同时点火，火势向内蔓延，向前推进的火线相遇处各点的轨迹就是中轴。

在 MATLAB 中可以调用 bwmorph 函数提取出图像的骨架或细化，实现代码见例程 10-10。

例程 10-10

```
I1= imread('circles.tif');
imshow(I1);
I2 = bwmorph(I1,'skel',Inf);
imshow(I2)
I3= bwmorph(I1,'thin',Inf);
imshow(I3)
```

处理结果如图 10-30 和图 10-31 所示。



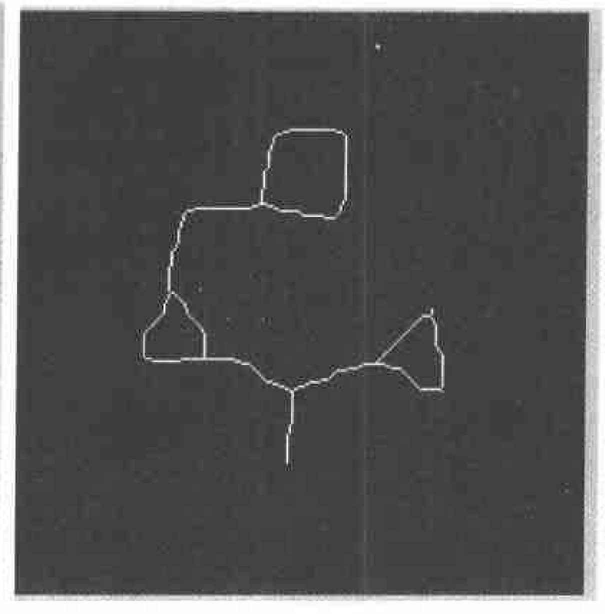




图 10-30　circles.tif 图像的骨架

图 10-31　circles.tif 图像细化的结果

比较两幅图可以更加清楚细化和提取骨架之间的差别。

10.3.6 查找表操作

在 MATLAB 中，有些二进制图像操作能够通过查找表非常容易地实现。一个查找表就是一个列矢量，其每个元素均表示邻域中的一种可能的像素组合的返回值。

利用图像处理工具箱中提供的 makelut 函数，我们可以为各种操作创建查找表 LUT。

makelut 函数可以为 2×2 和 3×3 的边沿创建查找表，分别如图 10-32 和图 10-33 所示。

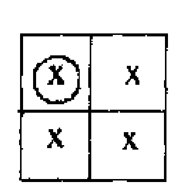


图 10-32　2×2 的边沿

X	X	X
X	(X)	X
X	X	X

图 10-33　3×3 的边沿

makelut 函数语法格式如下：

```
lut = makelut(fun,n)
lut = makelut(fun,n,P1,P2,…)
```

其中 fun 是一个包含了函数名的字符串或 inline 函数对象，此函数以 2×2 或 3×3 的 0-1 矩阵作为输入，返回一个标量。n 的值是 2 或 3，表明 fun 的输入值的尺寸。P1,P2,… 也是传递给 fun 的参数。返回值 lut 是一个 double 型的矢量。

makelut 函数生成 lut 的方法是：把 2×2 邻域或 3×3 邻域内的所有值都传递给 fun，每次一个，构造含 16 个元素的矢量（2×2 邻域）或含 512 个元素的矢量（3×3 邻域）。矢量中包含了 fun 对每个可能的近邻的输出。

makelut 函数通常与 applylut 函数联合使用，applylut 函数语法格式如下：

```
A = applylut(BW,lut)
```

其中 BW 是输入的二值图像，可以是 uint8 型或 double 型；lut 是由 makelut 返回的 16 元素的矢量，或 512 元素的矢量，可以是 uint8 型或 double 型；A 的值取决于 lut 中的值，如果 lut 中的值全部是 1 和 0，则 A 是二值图像。如果 lut 中的所有元素都是 0 到 255 之间的整数，则不管 lut 是什么类型的，A 都是 uint8 型的；否则是 double 型的。

applylut 对二值图像执行邻域操作，生成对 lut 的索引矩阵，然后用 lut 中的实际值替换索引值。具体的算法取决于所采用的邻域是 2×2 的，还是 3×3 的。

- 2×2 邻域

在这种情形下，lut 的长度为 16，每个邻域中都有 4 个像素，每个像素都有 2 个可能的状态，所以交换的总次数为：$2^4=16$。Applylut 将二值图像与矩阵

$$\begin{bmatrix} 8 & 2 \\ 4 & 1 \end{bmatrix}$$

进行卷积，生成索引矩阵。卷积结果中的值都是[0,15]范围内的整数值，applylut 使用卷积中与输入二值图像矩阵相同大小的中心部分，将每个值都加 1，使得范围变为[1,16]。然后，将索引矩阵中每个位置上的值用 lut 中索引值所指向的值替换，构造出 A。

- 3×3 邻域

在这种情形下，lut 的长度为 512，每个邻域中都有 9 个像素，每个像素都有 2 个可能的状态，所以交换的总次数为：$2^9=512$。Applylut 将二值图像与矩阵

$$\begin{bmatrix} 256 & 32 & 4 \\ 128 & 16 & 2 \\ 64 & 8 & 1 \end{bmatrix}$$

进行卷积，生成索引矩阵。卷积结果中的值都是[0, 511]范围内的整数值，applylut 使用卷积中与输入二值图像矩阵相同大小的中心部分，将每个值都加 1，使得范围变为[1,512]。然后，将索引矩阵中每个位置上的值用 lut 中索引值所指向的值替换，构造出 A。

下面举例，通过查找表来修改包含文本的图像 text.tif，如图 10-34 所示。

例程 10-11 通过查找表来修改包含文本的图像。

例程 10-11

```
%首先利用嵌入函数创建查找表
f=inline('sum(x(:)) == 4');
lut = makelut(f,2);
%然后调用 applylut 函数利用查找表 lut 对图像进行操作
I1 = imread('text.tif');
I2 = applylut(I1,lut);
imshow(I1)
figure, imshow(I2)
```

最后的结果如图 10-35 所示。

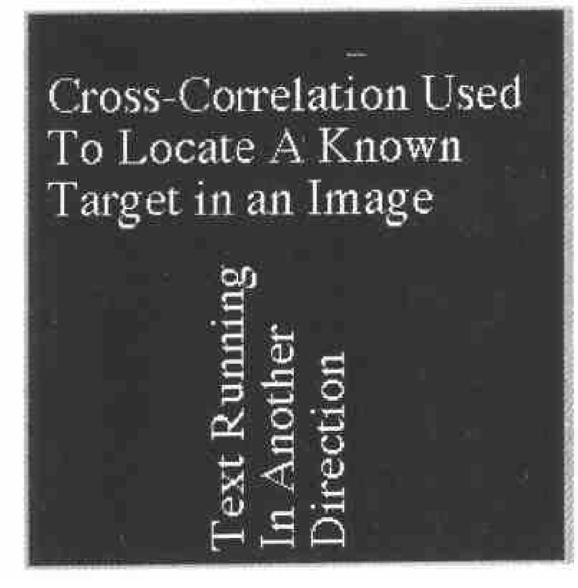


图 10-34 text.tif 图像

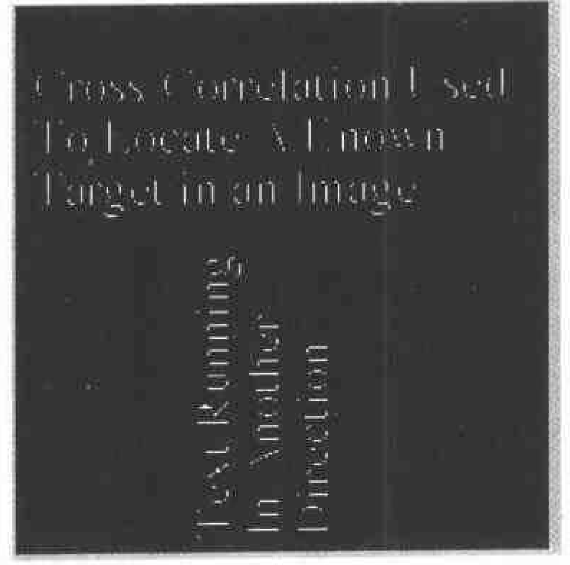


图 10-35 应用查找表的结果

本例使用了 2×2 邻域来执行腐蚀。当某输入像素的四个相邻像素的值都为 on 时，该像素的输出值才为 on。

10.3.7 区域填充

此二值图像操作与其他操作的不同之处在于它对背景像素进行操作，而不对前景像素

进行操作。在 MATLAB 中由 bwfill 函数来实现，语法格式如下：

```
BW2 = bwfill(BW1,c,r,n)
BW2 = bwfill(BW1,n)
[BW2,idx] = bwfill(…)
BW2 = bwfill(x,y,BW1,xi,yi,n)
[x,y,BW2,idx,xi,yi] = bwfill(…)
BW2 = bwfill(BW1,'holes',n)
[BW2,idx] = bwfill(BW1,'holes',n)
```

BW2 = bwfill(BW1,c,r,n) 对图像 BW1 进行区域填充，(r,c)为填充的起始坐标，如果是等长的矢量，则从(r(k),c(k))开始进行平行填充。n 为区域的连通数，可取值为 4 或 8，分别对应着前景的 4-邻域边沿约定和 8-邻域边沿约定，默认值为 8。BW1 的前景由所有值为 1 的像素组成。

BW2 = bwfill(BW1,n) 将图像 BW1 显示在屏幕上，由用户用鼠标交互地选取填充的起始点。如果忽略了 BW1，则函数对当前坐标轴内的图像进行操作。单击【常规】按钮来增加点，按下【Backspace】键或【Delete】键删除前面选取的点，按下【Shift】键并单击鼠标左键、鼠标右键或双击鼠标左键可以确定终点，并开始填充；按下【Enter】键，则终止选点。

[BW2,idx] = bwfill(…) 返回填充点的线性索引。

BW2 = bwfill(x,y,BW1,xi,yi,n) 用矢量 x，y 为 BW1 设置非默认的空间坐标系，xi 和 yi 是标量或等长的矢量，指定起始填充点在坐标系中的位置。

[x,y,BW2,idx,xi,yi] = bwfill(…) 返回的输出图像存放在 BW2 中，横纵数据分别返回到 x、y 中，所有填充像素点的线性索引返回到 idx 中，填充起始点返回到 xi、yi 中。

BW2 = bwfill(BW1,'holes',n) 自动确定哪些像素点在对象孔洞中，然后将这些像素点的值由 0 改为 1。n 的默认值为 8。

[BW2,idx] = bwfill(BW1,'holes',n)返回函数填充的所有像素点的线性索引。

如果函数没有输出值，则将填充的结果显示出来。特别要说明的是：当前景是 4-邻域边沿约定时，背景是 8-邻域边沿约定；反之亦然。输入图像可以是 double 型的或 uint8 型的，输出图像是 uint8 型的。

例程 10-12 用形态学的方法填充图像中的孔洞。

例程 10-12

```
%首先，对 blood1.tif 图像求反，使前景像素值变成 1
I = imread('blood1.tif');
BW1 = ~im2bw(I);
%然后，进行填充
BW2 = bwfill(BW1,'holes');
subplot(1,2,1),imshow(BW1)
subplot(1,2,2),imshow(BW2)
```

结果见图 10-36 和图 10-37。



图 10-36 blood1.tif 求反的图像

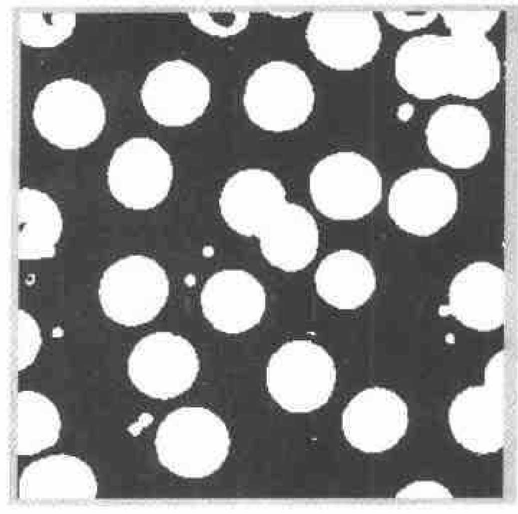


图 10-37 填充图像中的孔洞

10.3.8 对象提取

对象提取在模式识别等很多应用领域中都很有用，因此也是图像处理中的重要内容。在 MATLAB 中可以由 bwselect 函数来实现，语法格式如下：

```
BW2 = bwselect(BW1,c,r,n)
BW2 = bwselect(BW1,n)
[BW2,idx] = bwselect(…)
BW2 = bwselect(x,y,BW1,xi,yi,n)
[x,y,BW2,idx,xi,yi] = bwselect(…)
```

BW2 = bwselect(BW1,c,r,n)返回二值图像，其中包含了与像素（r,c）交叠的对象。r、c 可以是标量或等长的矢量；如果 r、c 是矢量，则 BW2 包含了与各像素(r(k),c(k))交叠的对象集。*n* 可以取 4 或 8，分别对应着对象的 4-邻域边沿约定和 8-邻域边沿约定，默认值为 8。对象由所有值为 1 的像素组成。

BW2 = bwselect(BW1,n) 将图像 BW1 显示在屏幕上，由用户用鼠标交互地选取（r,c）坐标。如果忽略了 BW1，则函数对当前坐标轴内的图像进行操作。单击【常规】按钮来增加点，按下【Backspace】键或【Delete】键删除前面选取的点，按下【Shift】键并单击鼠标左键、鼠标右键或双击鼠标左键可以确定终点，并开始填充；按下【Enter】键，则终止选点。

[BW2,idx] = bwselect(…)返回所选对象中像素点的线性索引。

BW2 = bwselect(x,y,BW1,xi,yi,n) 用矢量 x，y 为 BW1 设置非默认的空间坐标系，xi 和 yi 是标量或等长的矢量，指定了对象在坐标系中的位置。

[x,y,BW2,idx,xi,yi] = bwselect(…)返回的输出图像存放在 BW2 中，横纵数据分别返回到 x、y 中，所选对象中所有像素点的线性索引返回到 *idx* 中，所指定的空间坐标返回到 xi、yi 中。

如果函数没有输出值，则将填充的结果显示出来。输入图像可以是 double 型的或 uint8 型的，输出图像是 uint8 型的。

例程 10-13 以 text.tif 图像为例，提取文本图像中的字符对象。

例程 10-13

```
BW1 = imread('text.tif');
c = [16 90 144];
r = [85 197 247];
BW2 = bwselect(BW1,c,r,4);
imshow(BW1)
figure, imshow(BW2)
```

结果如图 10-38 和图 10-39 所示。

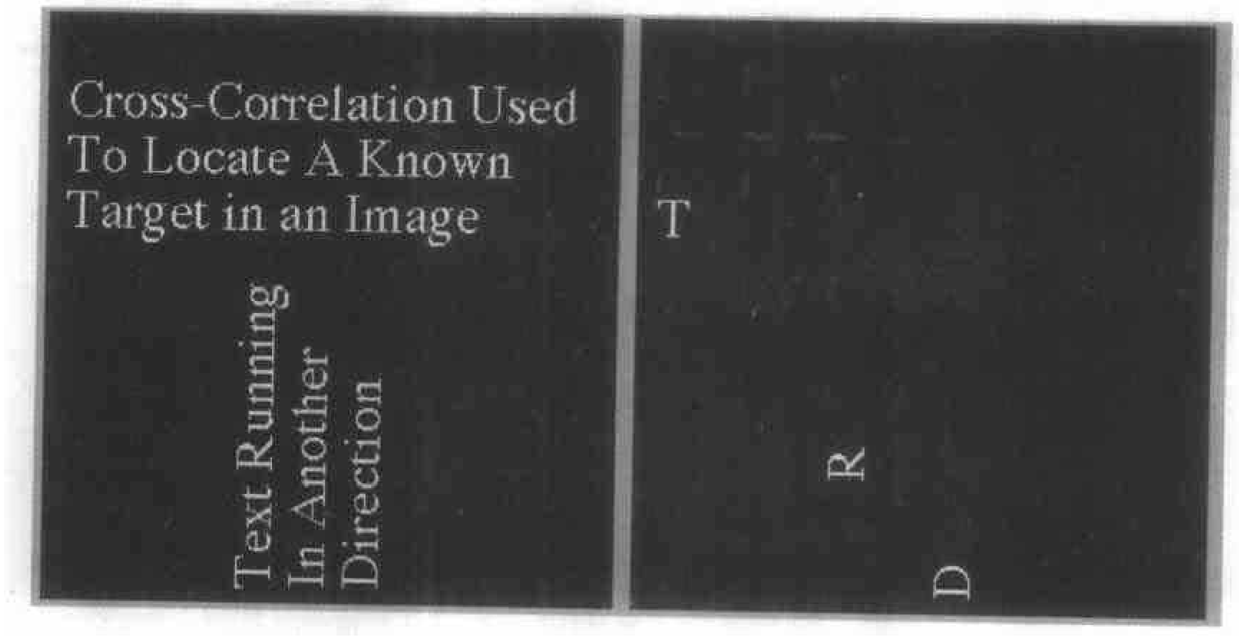


图 10-38 text.tif 图像　　图 10-39 提取字符对象

第 11 章　小波分析和分形几何用于图像处理的 MATLAB 实现

其他相关学科的发展，也在推动着图像处理技术不断前进。近些年，小波分析、分行几何及神经网络技术这些新技术在图像处理应用上都取得了一些新的进展。特别是小波分析技术，它克服了傅里叶变换的许多弱点，广泛应用于图像压缩中。国外一些公司已经将这种技术用于 Internet 环境的图像数据传输中，提供商业化的服务，这很好地解决了网络带宽不足和图像传输实时性的问题。本章着重介绍小波分析在图像处理上的应用，然后简要介绍分形几何用于图像处理的 MATLAB 实现方法。

本章主要内容：

- 小波分析引论
- 小波分析用于图像处理
- 小波包分析用于图像处理
- 分形几何的 MATLAB 实现

11.1　小波分析用于图像处理

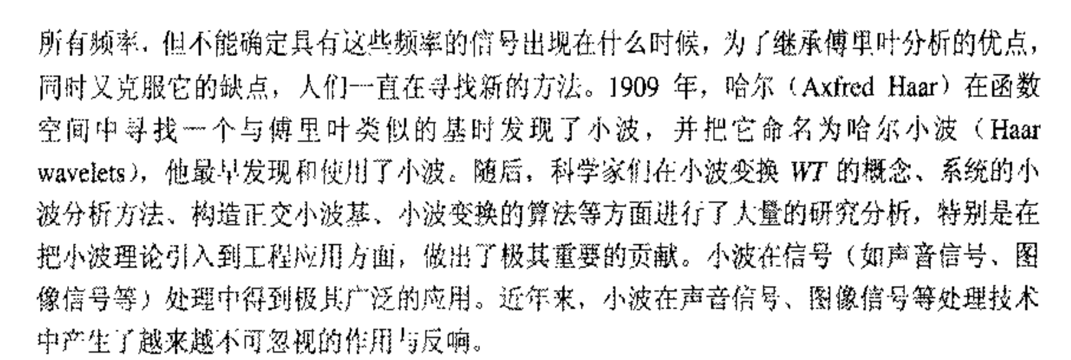
小波变换在数学领域本身的许多学科、信号分析、图像处理、量子力学、电子对抗、计算机识别、数据压缩、CT 成像、地震勘探数据处理、边缘检测、音乐与语音人工合成、机械故障诊断、大气与海洋波的分析、分形力学、流体湍流及天体力学方面都已取得了具有科学意义和应用价值的重要成果。除了微分方程的求解之外，原则上能用傅里叶分析的地方均可用小波分析，甚至能获得更好的结果。

小波分析在图像处理中已经得到广泛的应用，一般来说，主要体现在以下几个方面：图像的分解和重构、图像压缩、图像消噪、图像增强、图像平滑、图像融合等。

11.1.1　引言

小波是近十几年才发展起来并迅速应用到图像处理和语音分析等众多领域的一种数学工具，是继一百多年前的傅里叶分析之后的一个重大突破，无论是对古老的自然学科还是对新兴的高新技术应用学科都产生了强烈的冲击，是国际科技界和众多学术团体高度关注的前沿领域。

由傅里叶理论我们知道，一个信号可表示成一系列正弦和余弦函数之和。用傅里叶表示一个信号时，只有频率分辨率而没有时间分辨率，这就意味我们可以确定信号中包含的



所有频率，但不能确定具有这些频率的信号出现在什么时候，为了继承傅里叶分析的优点，同时又克服它的缺点，人们一直在寻找新的方法。1909 年，哈尔（Axfred Haar）在函数空间中寻找一个与傅里叶类似的基时发现了小波，并把它命名为哈尔小波（Haar wavelets），他最早发现和使用了小波。随后，科学家们在小波变换 *WT* 的概念、系统的小波分析方法、构造正交小波基、小波变换的算法等方面进行了大量的研究分析，特别是在把小波理论引入到工程应用方面，做出了极其重要的贡献。小波在信号（如声音信号、图像信号等）处理中得到极其广泛的应用。近年来，小波在声音信号、图像信号等处理技术中产生了越来越不可忽视的作用与反响。

1．小波概念

小波变换的定义是把某一被称为基本小波（也叫母小波 mother wavelet）的函数 $\psi(t)$ 做位移 τ 后，再在不同尺度 a 下与待分析的信号 $x(t)$ 做内积：

$$WT_x(a,\tau)=\frac{1}{\sqrt{a}}\int_{-\infty}^{\infty}x(t)\psi^*(\frac{t-\tau}{a})\mathrm{d}t\,,\quad a>0$$

等效的频域表示是：

$$WT_x(a,\tau)=\frac{\sqrt{a}}{2\pi}\int_{-\infty}^{\infty}X(\omega)\Psi^*(a\omega)\mathrm{e}^{+j\omega\tau}\mathrm{d}\omega$$

式中 $X(\omega)$ 和 $\Psi(\omega)$ 分别是 $x(t)$ 和 $\psi(t)$ 的傅里叶变换。

小波变换有以下特点：

1）有多分辨率（multi-resolution），也叫多尺度（multi-scale）的特点，可以由粗及细地逐步观察信号。

2）可以看成用基本频率特性为 $\Psi(\omega)$ 的带通滤波器在不同尺度 a 下对信号做滤波。由于傅里叶变换的尺度特性可知，这组滤波器具有品质因数恒定，即相对带宽（带宽与中心频率之比）恒定的特点。注意，a 越大相对频率越低。

3）适当地选择基小波，使 $\psi(t)$ 在时域上为有限支撑，$\Psi(\omega)$ 在频域上也比较集中，就可以使 *WT* 在时、频域都具有表征信号局部特征的能力，因此有利于检测信号的瞬态或奇异点。

正如上所述，小波分析的一个主要优点就是能够分析信号的局部特征。比如说，采用小波分析可以发现叠加在一个非常规范的正弦信号上的一个非常小的畸变信号的出现时间。传统的傅里叶变换只能得到平坦的频谱上的两个尖峰。利用小波分析可以非常准确地分析出信号在什么时刻发生畸变。小波分析可以检测出许多其他分析方法忽略的信号特性，例如，信号的趋势、信号的高阶不连续点、自相似特性。小波分析还能以非常小的失真度实现对信号的压缩与消噪，它在图像数据压缩方面的潜力已经得到确认。在二维情况下，小波分析除了“显微”能力外还具有“极化”能力（即方向选择性），因而引人注意。

2．小波在图像处理中的应用

图像处理是小波分析应用的重要领域，近年来小波分析已被证明是进行图像处理强有力

的工具之一。由于小波分析技术可以将信号或图像分层次按小波基展开，并且可以根据图像的性质及事先给定的图像处理要求确定到底要展开到哪一级为止，从而不仅能有效地控制计算量，满足实时处理的需要，而且可以方便地实现通常由子带编码技术实现的累进编码（即采取逐步浮现的方式传送多媒体图像）。同时，小波变换具有放大、缩小和平移的功能，能够很方便地产生各种分辨率的图像，从而适合于不同分辨率图像的 I/O 设备和不同传输速率的通信系统。

由于小波变换的突出优点，所以在最近颁布的运动图像压缩标准 MPEG 4 中的视觉纹理模式就支持纹理和静态图像编码。这种模式基于零高度树小波算法，在非常宽的比特率范围内具有很高的编码效率。除了具有很高的压缩效率外，它还提供了空间和质量的可缩放性，以及对任意形状目标的编码。因此下面将结合 MATLAB 6.5 语言介绍小波分析在图像处理这一领域的应用。

11.1.2　基于小波的图像分解与重构

如图 11-1 所示是图像的小波分解示意图，L 表示低频，H 表示高频，下标 1、2 表示一级或二级分解。分解的数据传递示意图如图 11-2 所示（图中“↓2”表示下采样）。图 11-3 是重构的数据传递示意图（图中“↑2”表示上采样）。

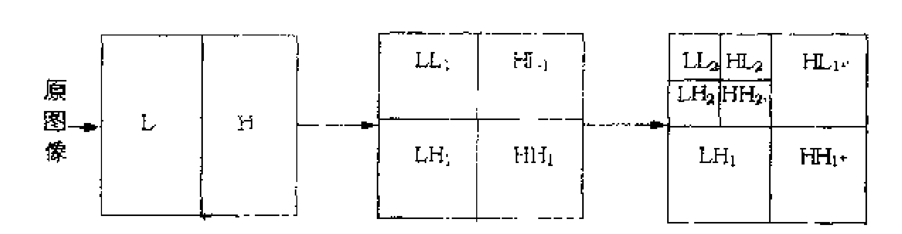


图 11-1　图像的小波分解示意图

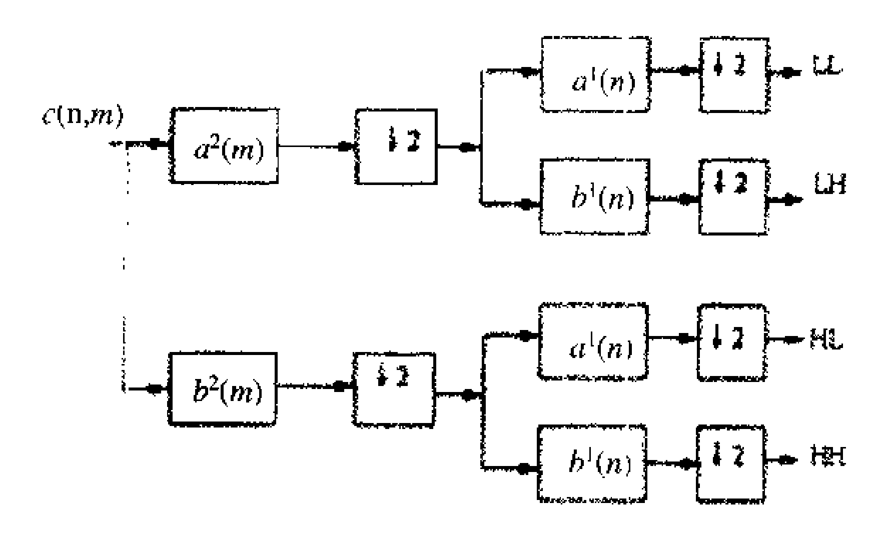


图 11-2　小波分解数据流示意图

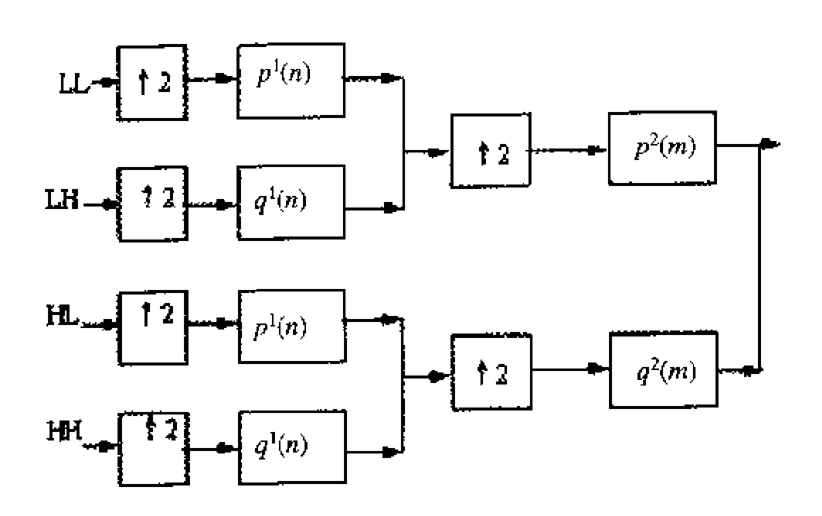


图 11-3 小波重构数据流示意图

下面以例子的形式对基于小波的图像分解与重构进行介绍。

1、图像的单尺度分解

首先我们需要将对图像进行加载，其命令行如下：

```
load sinsin;

whos
  Name        Size              Bytes   Class
  X           256×256           double array
  map         192×3             4608    double array
image(X);
colormap(map);
colorbar;
```

显示的初始图像如图 11-4 所示。

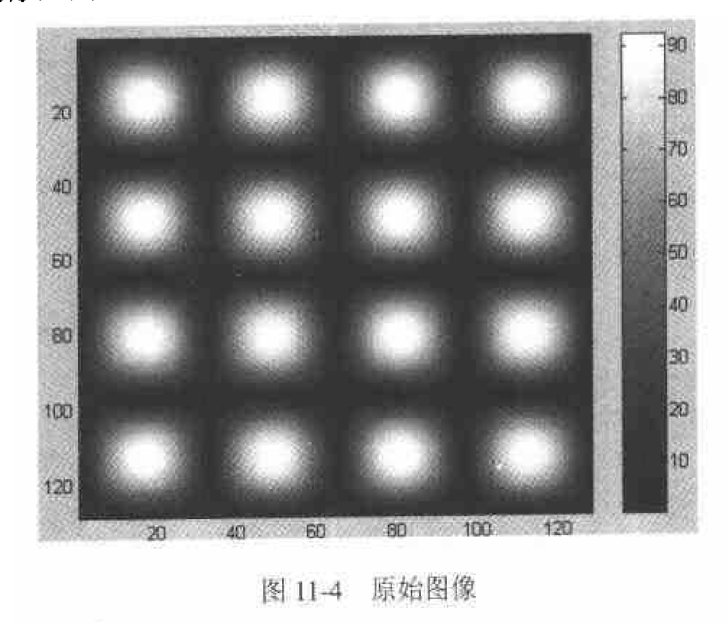
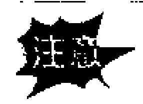


图 11-4 原始图像



如果图像的色彩条是光滑的，那么小波变换就可以直接对其进行分解；否则需将图像转换成为灰度级索引图像方可进行分解。

例程11-1是图像单尺度分解的MATLAB程序。

例程11-1

```
[cA1,cH1,cV1,cD1]=dwt2(X,'bior3.7');

%从分解系数中提取近似和细节
[cA1,cH1,cV1,cD1]=dwt2(X,'bior3.7');
A1=upcoef2('a',cA1,'bior3.7',1);
H1=upcoef2('h',cH1,'bior3.7',1);
V1=upcoef2('v',cV1,'bior3.7',1);
D1=upcoef2('d',cD1,'bior3.7',1);

%显示近似和细节
figure(2);
colormap(map);
subplot(2,2,1);
image(wcodemat(A1,192));
title('近似 A1');
subplot(2,2,2);
image(wcodemat(H1,192));

title('水平细节 H1');
subplot(2,2,3);
image(wcodemat(V1,192));
title('垂直细节 V1');
subplot(2,2,4);
image(wcodemat(D1,192));
title('对角细节 D1');
```

结果如图11-5所示。

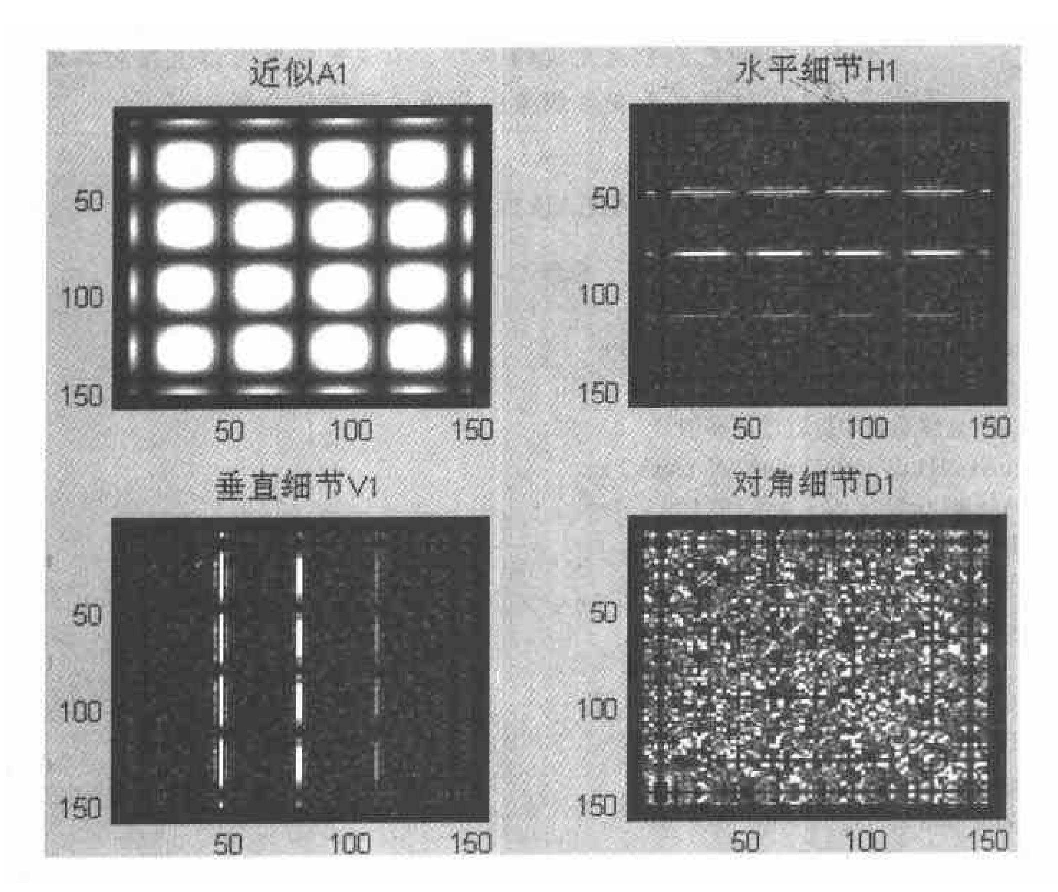


图 11-5　单尺度分解结果

2. **图像的多尺度分解**

例程 11-2 是图像多尺度分解的 MATLAB 程序。

结果如图 11-6 所示。为了验证多尺度分解的正确性，我们对分解后的图像进行了重构，如图 11-7 所示。

例程 11-2

```
[C,S]=wavedec2(X,2,'bior3.7');

%提取分解后的近似和细节系数
cA2=appcoef2(C,S,'bior3.7',2);
[cH2,cV2,cD2]=detcoef2('all',C,S,2);
[cH1,cV1,cD1]=detcoef2('all',C,S,2);

%从系数 C 重构第二层近似
A2=wrcoef2('a',C,S,'bior3.7',2);

%从系数 C 重构第一、二层细节
H1=wrcoef2('h',C,S,'bior3.7',1);
V1=wrcoef2('v',C,S,'bior3.7',1);
D1=wrcoef2('d',C,S,'bior3.7',1);
H2=wrcoef2('h',C,S,'bior3.7',2);
```

```
V2=wrcoef2('v',C,S,'bior3.7',2);
D2=wrcoef2('d',C,S,'bior3.7',2);

%显示多尺度分解的结果;
colormap(map);
subplot(2,4,1);
image(wcodemat(A1,192));
title('近似 A1');

subplot(2,4,2);
image(wcodemat(H1,192));
title('水平细节 H1');

subplot(2,4,3);
image(wcodemat(V1,192));
title('垂直细节 V1');

subplot(2,4,4);
image(wcodemat(D1,192));
title('对角细节 D1');

subplot(2,4,5);
image(wcodemat(A2,192));
title('近似 A2');

subplot(2,4,6);
image(wcodemat(H2,192));
title('水平细节 H2');

subplot(2,4,7);
image(wcodemat(V2,192));
title('垂直细节 V2');

subplot(2,4,8);
image(wcodemat(D2,192));
title('对角细节 D2');
%从多尺度分解后的系数重构原始图像并显示结果

figure(3);
X0=waverec2(C,S,'bior3.7');
image(X0);
colormap(map);
colorbar;
```

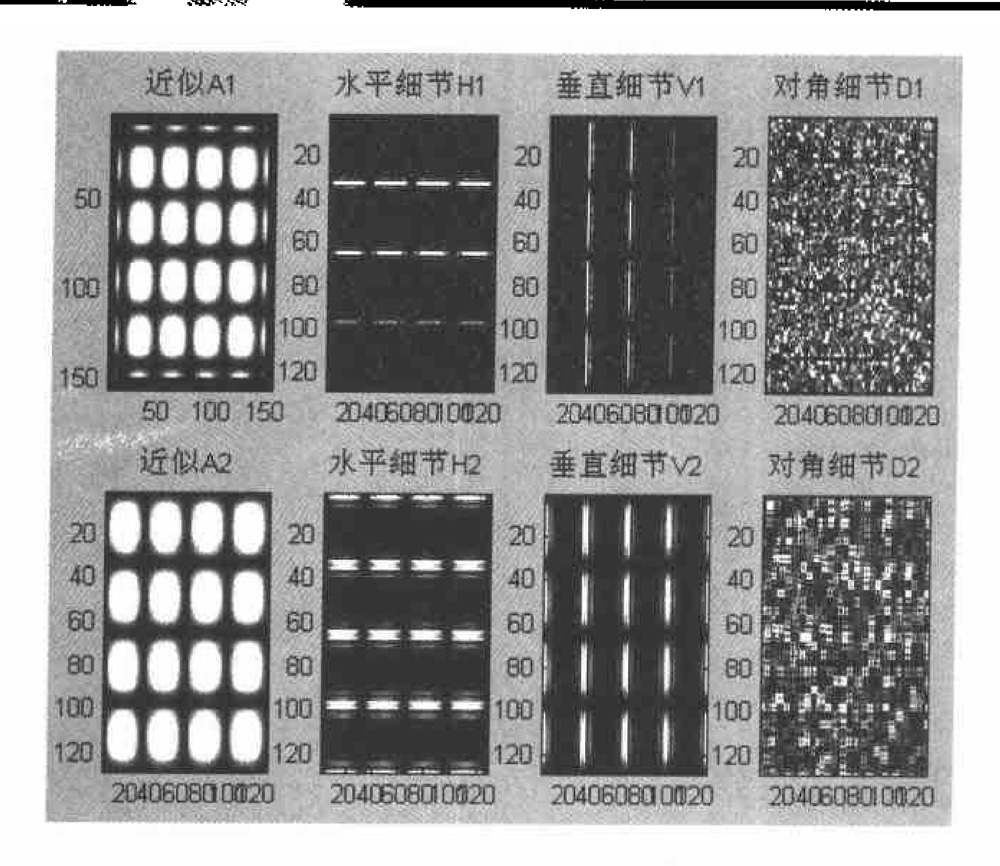


图 11-6　多尺度分解后的近似和细节

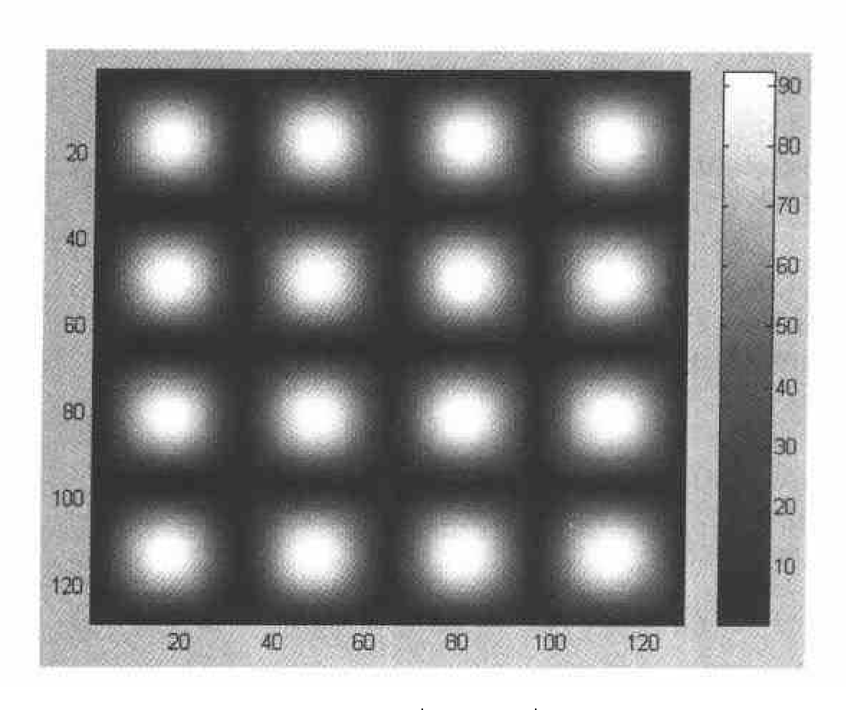


图 11-7　重构后的图像

将图 11-7 和图 11-4 进行比较可以看出，重构后的图像与原始图像非常相似，因此二层多尺度分解并没有带来很大的图像失真。通过这个例子，可以直观感受到小波在图像压缩中的突出优势。

11.1.3 基于小波的图像消噪

1. 小波分析进行二维图像消噪

二维小波分析用于图像消噪主要有 3 个步骤：

（1）二维图像信号的小波分解。

在这一步，应当选择合适的小波和恰当的分解层次（记为 N），然后对待分析的二维图像信号 X 进行 N 层分解计算。

（2）对分解后的高频系数进行阈值量化。

对于分解的每一层，选择一个恰当的阈值，并对该层高频系数进行软阈值量化处理。在此，阈值选取规则同前面的信号处理部分。

（3）二维小波的重构图像信号。

同样，根据小波分解后的第 N 层近似（低频系数）和经过阈值量化处理后的各层细节（高频系数），来计算二维信号的小波重构。

在这三步中，重点内容仍然是如何选取阈值和如何进行阈值量化这个步骤。

例程 11-3 利用小波分析进行图像消噪。

例程 11-3

```
%装载原始图像信号并显示
load wifs;
subplot(2,2,1);
image(X);
colormap(map);
title('原始图像');
axis square;

%生成含噪图像并图示
init=2055615866;
rand('seed',init);
XX=X+randn(size(X));
subplot(2,2,2);
image(XX);
colormap(map);
title('含噪图像');
axis square;

%下面对图像消噪
%首先用 sym4 小波函数对 XX 进行 2 层分解
[c,l]=wavedec2(XX,2,'sym4');

%实现低通滤波消噪
a1=wrcoef2('a',c,l,'sym4',1);
```

```
%再次实现低通滤波消噪
a2=wrcoef2('a',c,l,'sym4',2);

%图示消噪处理后的结果
subplot(2,2,3);
image(a1);
colormap(map);
title('第一次消噪图像');
axis square;
subplot(2,2,4);
image(a2);
colormap(map);
title('第二次消噪图像');
axis square;
```

图像消噪的结果如图 11-8 所示。

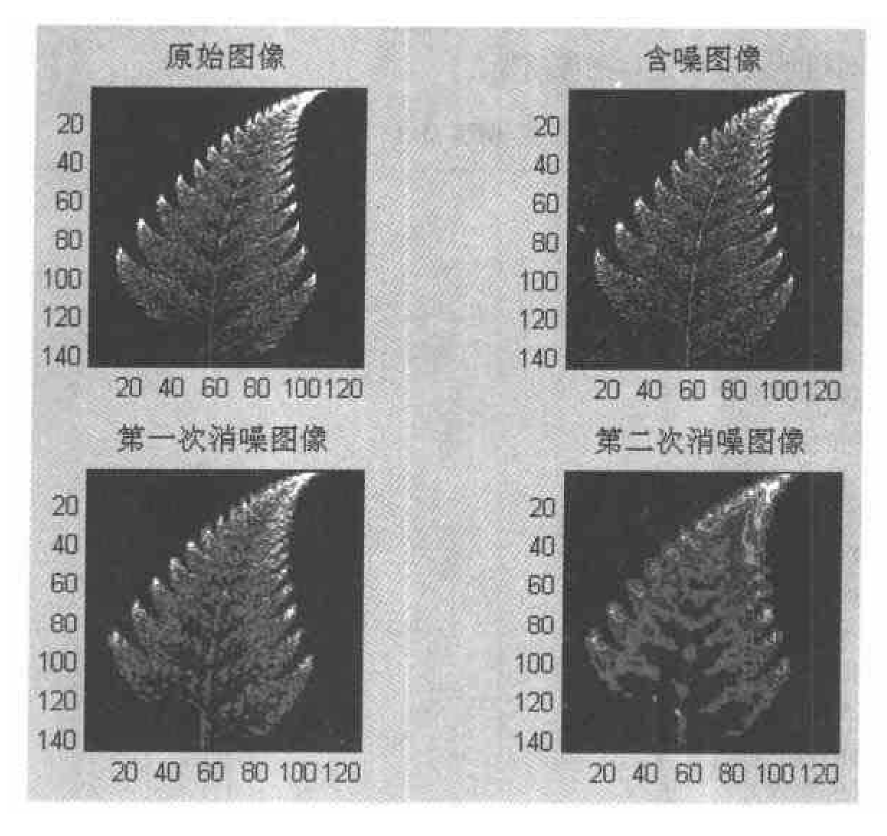


图 11-8　例程 11-3 的运行结果



二维图像信号的消噪步骤有三步，和一维信号的消噪步骤完全相同。所不同的是，处理时用二维小波分析工具代替了一维小波分析工具。因此，对二维图像信号的消噪方法也同样地适合于一维信号，尤其对几何图形更为适合。



2．小波包分析进行图像消噪

在这里，小波包分析进行图像消噪与压缩处理的基本原理和方法与对信号消噪与压缩的相同，请读者朋友参看《MATLAB 6.5 辅助小波分析与应用》一书第 6 章的相关内容。这里仅以具体的实例来说明小波包分析在图像消噪处理中的应用。

例程 11-4 是小波包分析进行图像消噪的 MATLAB 程序。

例程 11-4

```
%装载并显示原始图像
load wifs;
subplot(2,2,1);
image(X);
colormap(map);
title('原始图像');
axis square;

%生成含噪图像并图示
init=2055615866;
randn('seed',init);
X1=X+10*randn(size(X));
subplot(2,2,2);
image(X1);
colormap(map);
title('含噪图像');
axis square;

%消噪处理：设置函数 wpdencmp 的消噪参数
thr=10;sorh='s';
crit='shannon';
keepapp=0;
X2=wpdencmp(X1,sorh,3,'sym4',crit,thr,keepapp);

%画出消噪后的图像
subplot(2,2,3);
image(X2);
colormap(map);
title('全局阈值消噪图像');
axis square;
%对图像进行平滑处理以增强消噪效果，这里采用中值滤波的方法
for i=2:175;
    for j=2:259
        Xtemp=0;
        for m=1:3
            for n=1:3
                Xtemp=Xtemp+X2(i+m-2,j+n-2);
            end
```

```
            end
            Xtemp=Xtemp/9;
            X3(i,j)=Xtemp;
        end
    end

    %图示平滑结果
    subplot(2,2,4);
    image(X3);
    colormap(map);
    title('平滑后的图像');
    axis square;
    axis square;
```

结果如图 11-9 所示。

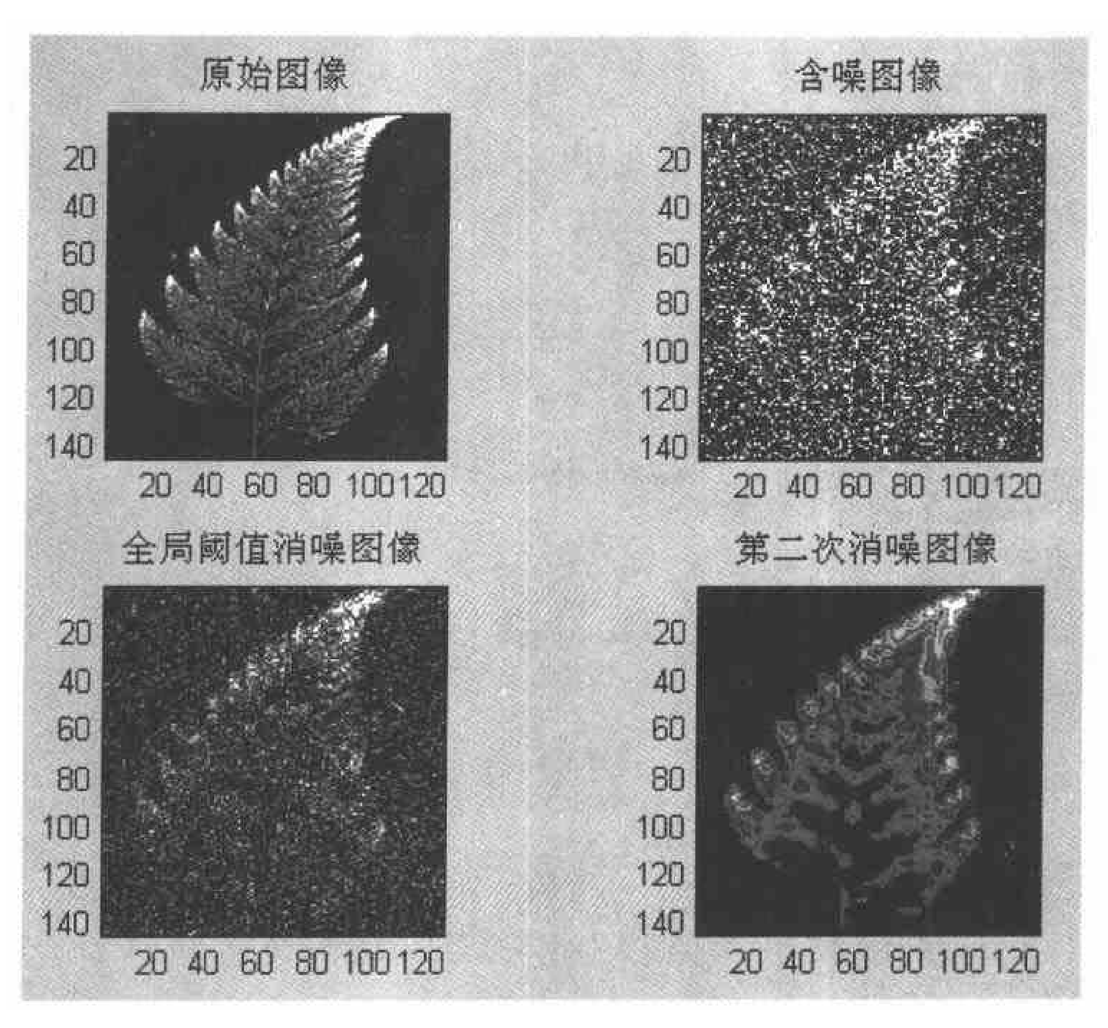


图 11-9　例 11-4 的运行结果

3．**总结和讨论**

传统的基于傅里叶变换的信号处理方法要求信号和噪声的频带重叠部分尽可能地小，

这样才能在频域中通过不变滤波的方法将信号和噪声区分开；而当它们的频域重叠时，这种方法就无能为力了。另外，如果采用线性小波变换和其他时间频率（或时间—尺度）法，通过选择不同基的方法可以使在相应坐标系内的信号和噪声的重叠尽可能地小，从而进行信噪分离。但是对大多数信号来说，合适的基的选择就是一个难题，因此这种方法的应用也受到了限制。

通过以上的介绍，小波变换克服了传统方法的不足。由于小波变换具有一种“集中”的能力，可以使一个信号的能量在小波变换域集中于少数系数上，再对小波系数进行阈值处理可以在小波变换域中除去低幅度的噪声和我们所不期望的信号，最后进行逆离散小波变换即可恢复原信号。尽管所恢复的信号丢失了一点细节，但是仍能恢复我们所期望的信号。当然这种方法不仅可以用于信号消噪，在数据压缩和分离信号方面也极其有用

最后，我们对信号消噪和压缩处理中所用到的软阈值和硬阈值做一简单讨论。在消噪和压缩处理中，如果采用软阈值处理方法，则估计信号和原始信号具有同样的光滑性，这是因为小波是一类光滑函数的无条件基，并且软阈值处理保证满足缩小条件，正是缩小条件保证了估计函数和原始函数具有相同的光滑性。此外，软阈值处理方法是满足缩小条件的最优估计。这种光滑性保证了软阈值处理方法不同于硬阈值或其他常用的频域处理方法，它所得到的估计信号同原始信号相比不会产生附加振荡。和传统的方法相似，阈值处理方法也会造成小程度上的图像（或数据）细节丢失，因此，在使用该方法时，还要考虑抑制噪声与所保留图像细小变化之间的权衡问题。从范数误差最小观点来说，硬阈值处理方法优于软阈值方法。但是由硬阈值处理得到的估计信号通常会有许多不期望的振荡，并且不具有期望的同原始信号相同的光滑性。

11.1.4 基于小波的图像压缩

图像数据往往存在各种信息的冗余，如空间冗余、信息熵冗余、视觉冗余和结构冗余等，因此有必要对它进行压缩。如果需要进行快速或实时传输及大量存储，在同等通信容量下，把图像数据压缩后再传输，就可以传输更多的图像信息，也就可以提高通信能力。小波分析用于图像压缩具有压缩比高、压缩速度快、压缩后能保持图像的特征基本不变的特点，且在传递过程中可以抗干扰。

小波分析进行图像压缩的基本原理是：根据二维小波分解算法，一幅图像做小波分解后，可得到一系列不同分辨率的图像，而表现一幅图像最主要的部分是低频部分，如果去掉图像的高频部分而只保留低频部分，则可以达到图像压缩的目的。基于小波分析的图像压缩方法有很多，比较成功的有小波包最优基方法、小波域纹理模型方法、小波变换零树压缩、小波变换向量量化压缩等。

虽然图像的数据是非常巨大的，但是邻近像素的灰度（将其看成随机变量）往往是高度相关的。因而可以采用适当的坐标变换去除相关，从而达到压缩数据的目的。

下面举几个图像压缩的实例。

1. 小波分析进行图像压缩

例程 11-5 是采用小波分析进行图像压缩的 MATLAB 程序。



对图像做小波分解后，可得到一系列不同分辨率的子图像（它们所对应的频率不相同）。而对于图像来说，表征它的最主要部分是低频部分，而高频部分大部分点的数值均接近于 0，而且频率越高，这种现象越明显。因此，利用小波分解去掉图像的高频部分而仅保留图像的低频部分是一种最简单的图像压缩方法。

例程 11-5

```
%装载并显示原始图像
load tartan;
subplot(2,2,1);
image(X);colormap(map);
title('原始图像 ');
axis square;
disp('压缩前图像的大小：');
whos('X')

%分解图像，提取分解结构中的第一层系数
[c,l]=wavedec2(X,2,'bior3.7');
cA1=appcoef2(c,l,'bior3.7',1);
cH1=detcoef2('h',c,l,1);
cD1=detcoef2('d',c,l,1);
cV1=detcoef2('v',c,l,1);

%重构第一层系数
A1=wrcoef2('a',c,l,'bior3.7',1);
H1=wrcoef2('h',c,l,'bior3.7',1);
D1=wrcoef2('d',c,l,'bior3.7',1);
V1=wrcoef2('v',c,l,'bior3.7',1);
c1=[A1 H1;V1 D1];
%图示第一层各频率信息
subplot(2,2,2);
image(c1);
title('分解后的低频和高频信息');
axis square;

%对图像进行压缩：保留第一层低频信息并对其进行量化编码
ca1=wcodemat(cA1,440,'mat',0);
ca1=0.5*ca1;
subplot(2,2,3);
image(ca1);
colormap(map);
title('第一次压缩图像');
axis square;
disp('第一次压缩图像的大小：');
whos('ca1')
```

```
%压缩图像：保留第二层低频信息并对其进行量化编码
cA2=appcoef2(c,l,'bior3.7',2);
ca2=wcodemat(cA2,440,'mat',0);
ca2=0.5*ca2;
subplot(2,2,4);
image(ca2);
colormap(map);
title('第二次压缩图像');
axis square;
disp('第二次压缩图像大小：');
whos('ca2')
```

压缩后的图像如图 11-10 所示。

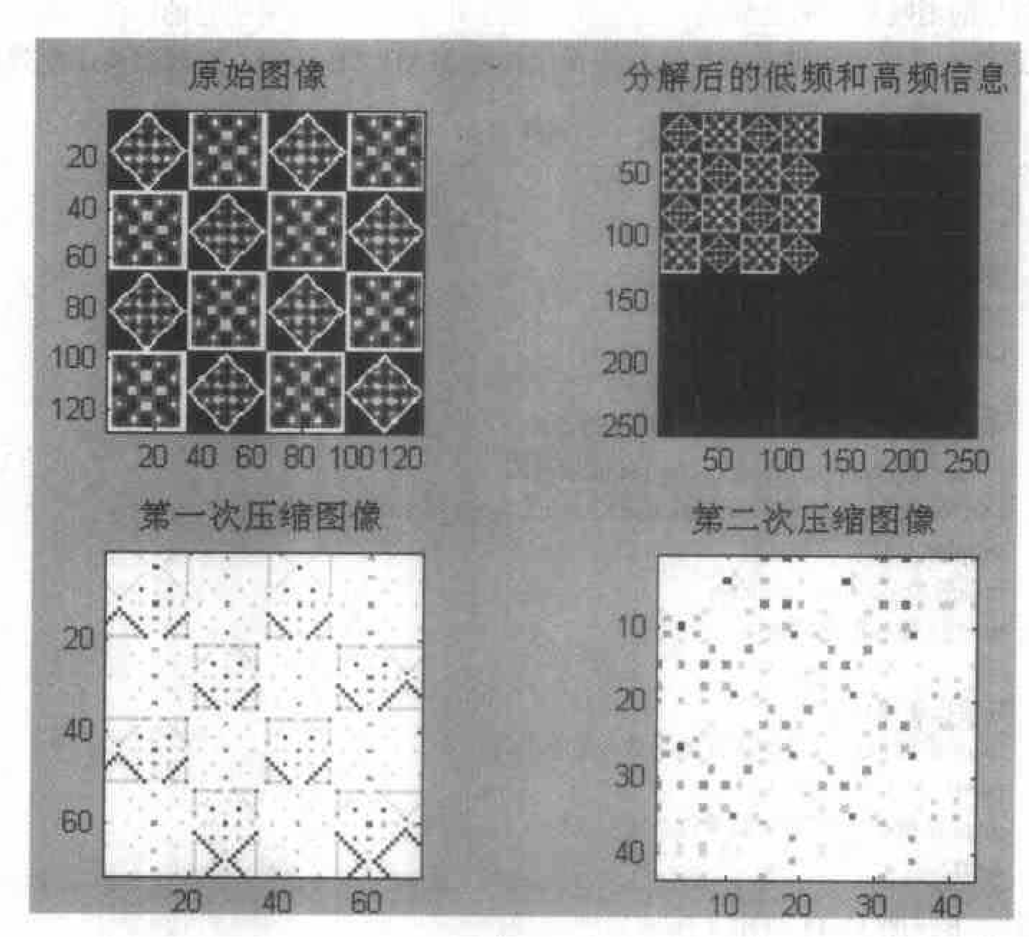


图 11-10　例程 11-5 的运行结果

上面的例程同时显示了压缩前后图像的大小信息。

压缩前图像的大小：

```
Name        Size          Bytes    Class
X          128x128        131072   double array
Grand total is 16384 elements using 131072 bytes
```

第一次压缩图像的大小：

```
Name        Size          Bytes   Class
ca1         71x71         40328   double array
Grand total is 5041 elements using 40328 bytes
```

第二次压缩图像大小：

```
Name        Size          Bytes   Class
ca2         43x43         14792   double array
Grand total is 1849 elements using 14792 bytes
```

从结果中可以看出，随着分解层数的增加，压缩比是递减的。

在 MATLAB 的小波工具箱中，有专门用于压缩处理的函数 wdencmp，它使得图像压缩过程更简单，这里不再介绍。

2．小波包分析进行图像压缩

在 MATLAB 的小波工具箱中，提供了一个函数 wpdencmp，这个函数是专门用小波包进行信号的消噪和压缩处理的。这里我们通过具体的例子来说明它的压缩功能。

例程 11-6 是采用小波包分析进行图像压缩的 MATLAB 程序，并观察其压缩效率。

例程 11-6

```
%装载并显示原始图像
load sinsin;
subplot(1,2,1);
image(X);colormap(map);
title('原始图像');

%采用默认的全局阈值进行图像压缩处理
[thr,sorh,keepapp,crit]=ddencmp('cmp','wp',X);
[Xc,treed,perf0,perfl2]=wpdencmp(X,sorh,3,'bior3.1',crit,thr,keepapp);
subplot(1,2,2);
image(Xc);colormap(map);
title('压缩后的图像');

%给出压缩效率
disp('小波分解系数中置 0 的系数个数百分比：');
perf0
disp('压缩后图像剩余能量百分比：');
perfl2
```

结果如下，图 11-11 给出了压缩后的图像。

小波分解系数中置 0 的系数个数百分比：

```
perf0 =
   86.3811
```

压缩后图像剩余能量百分比：

```
perfl2 =
   99.9957
```

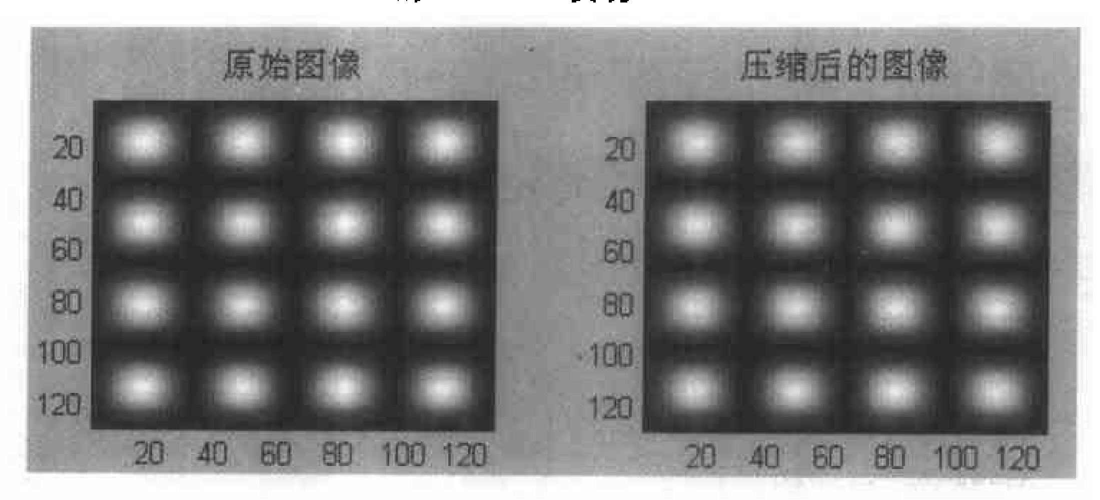


图 11-11　例程 11-6 的运行结果

3．**总结与讨论**

从以上的两个实例可以知道，应用小波分解进行图像压缩时，在理论上可以获得任意压缩比的压缩图像，且实现起来也较为简单。而利用小波包分析进行图像压缩时，随着所选用的分解小波、分解层次和量化阈值的不同所得结果不同，且实现起来也很简单。因此，小波变换在图像的压缩方面显示出了它的特性，即为图像从空间域变换到时间域提供了一种非常有效的方法，其作用与专门用于图像压缩的离散余弦变换（DCT）、傅里叶变换等类似。当然任何方法既有区别于其他方法的优点，同时又具有一定的缺陷，因此，要很好地对图像进行压缩处理，需要综合利用多种技术进行。这对基于小波分析的图像压缩也不例外，往往也需要利用小波分析和其他相关技术的有机结合才能达到较为完美的结果。

11.1.5　基于小波的图像增强

小波变换将一幅图像分解为大小、位置和方向均不相同的分量。在做逆变换之前，可根据需要对不同位置、不同方向上的某些分量改变其系数的大小，从而使得某些感兴趣的分量被放大而使某些不需要的分量减小。

下面我们用具体的例子来说明图像的增强。

例程 11-7 利用小波分析对所给图像进行增强处理。

例程 11-7

```
%装载并显示原始图像
load tartan;
subplot(2,2,1);
image(X);colormap(map);
title('原始图像');

%对图像 X 用小波 db3 进行 2 层分解
[c,l]=wavedec2(X,2,'db3');
Csize=size(c);
```

```
%对分解系数做处理以突出所需部分并弱化不需要部分
for i=1:Csize(2)
    if(c(i)>300)
        c(i)=2*c(i);
    else
        c(i)=0.5*c(i);
    end
end

%重构图像并显示
X1=waverec2(c,l,'db3');
subplot(2,2,2);
image(X1);colormap(map);
title('增强图像');
```

结果如图 11-12 所示。

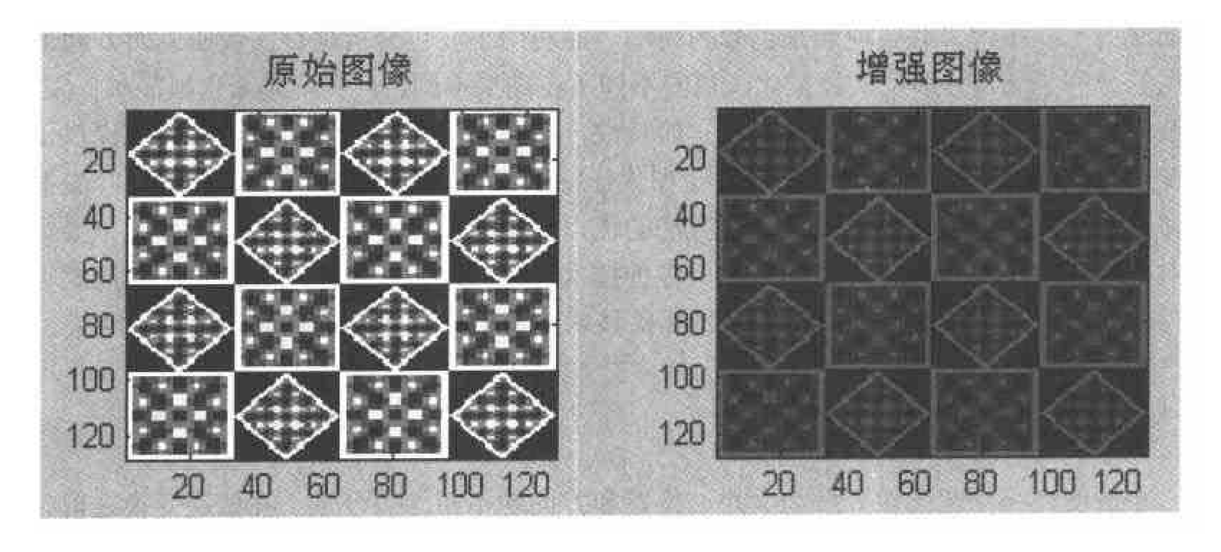


图 11-12 例程 11-7 的运行结果

分解后的图像，其主要信息（即轮廓）由低频部分来表征，而其细节部分则由高频部分表征。因此，在上述的例子中，我们对分解后的低频系数加权进行增强，而对高频部分加权进行弱化，经过如此处理后，即达到了增强图像的目的。

11.1.6 基于小波的图像平滑

例程 11-8 利用二维小波分析和图像的中值滤波对一给定的含噪图像进行平滑处理。

例程 11-8

```
%装载原始图像
load gatlin;

%对图像加噪声并显示出含噪图像
```

```
init=2788605800;
randn('seed',init);
X=X+10*randn(size(X));
subplot(1,2,1);
image(X);colormap(map);
title('含噪图像');

%图像平滑：应用中值滤波进行处理
for i=2:479
    for j=2:639
        Xtemp=0;
        for m=1:3
            for n=1:3
                Xtemp=Xtemp+X(i+m.2,j+n.2);
            end
        end
        Xtemp=Xtemp/9;
        X1(i,j)=Xtemp;
    end
end

%显示结果
subplot(1,2,2);
image(X1);colormap(map);
title('平滑图像');
```

结果如图 11-13 所示。

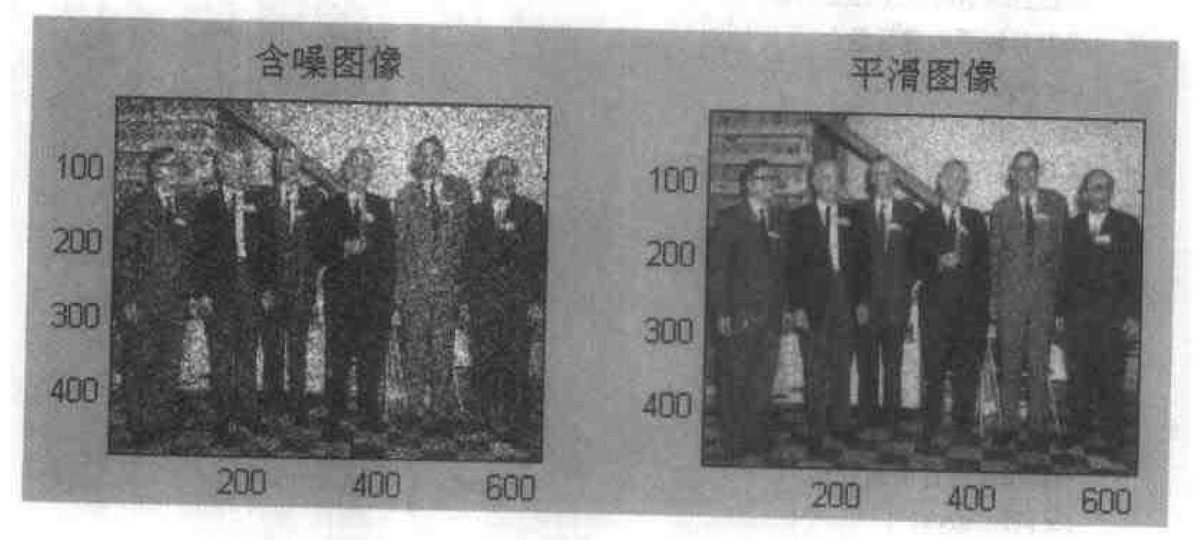


图 11-13 例程 11-8 的运行结果

在例程 11-8 中，我们首先对图像加了一个较大的白噪声，然后应用中值滤波对含噪图像进行处理。从图 11-13 可以看出，经过中值滤波处理后的含噪图像具有较好的平滑效果。

11.1.7 基于小波的图像融合

图像融合是将同一对象的两个或更多个图像合成在一幅图像之中，以便于满足人们的某种需要。这一技术应用于多频谱图像理解和医学图像处理等领域，使得融合图像更容易为人们所理解。

例程 11-9 是利用二维小波分析将图像 tartan.mat 和 sinsin .mat 融合在一起的 MATLAB 代码。

例程 11-9

```
%装载原始图像（tartan.mat 和 sinsin.mat）并显示
load tartan;
X1=X;map1=map;
subplot(2,2,1);
image(X1);colormap(map1);
title('图像 tartan');
load sinsin;
X2=X;map2=map;
subplot(2,2,2);
image(X2);colormap(map2);
title('图像 sinsin');

%对上述二图像进行分解
[c1,l1]=wavedec2(X1,2,'sym4');
[c2,l2]=wavedec2(X2,2,'sym4');
%对分解系数进行融合
c=c1+c2;

%应用融合系数进行图像重构并显示
XX=waverec2(c,l1,'sym4');
subplot(2,2,3);
image(XX);
title('融合图像 1');
Csize1=size(c1);

%对图像进行增强处理
for i=1:Csize1(2)
    c1(i)=1.2*c1(i);
end
Csize2=size(c2);
for j=1:Csize2(2)
    c2(j)=0.8*c2(j);
end
```

```
%通过减小融合系数以减小图像的亮度
c=0.5*(c1+c2);

%对融合系数进行图像重构
XXX=waverec2(c,l2,'sym4');

%显示重构结果
subplot(2,2,4);
image(XXX);
title('融合图像 2');
```

结果如图 11-14 所示。

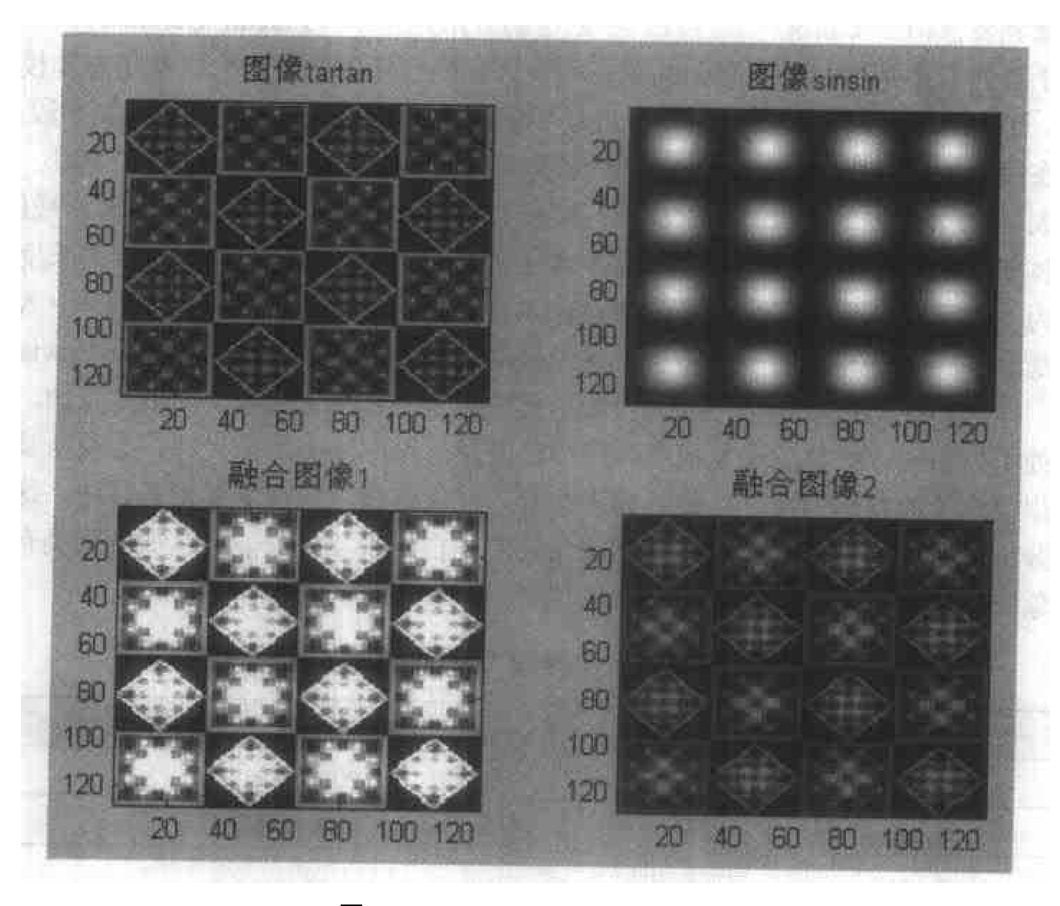


图 11-14　例程 11-9 的运行结果

11.2　分形几何的 MATLAB 实现

分形理论为描述事物的复杂性提供了有力的数学模型、定量参数和思维方法。目前分形几何已经渗透到数学、物理、化学、天文及心理、社会学等许多应用领域，它使人们对自然界和人类社会的认识提高到一个崭新的高度。近年来，分形理论与计算机技术结合后发展非常迅速，已经成为一门相当活跃的边缘学科，其理论研究和应用已经深入到人类活

动的方方面面，并取得了令人瞩目的成就。

分形理论在图像处理中也得到了广泛的应用，如模拟自然景物生成、图像分割、纹理特征提取、纹理分类和图像编码压缩等。

由于分形几何的历史不长，且仍在不断的发展中，所以 MATLAB 目前还没有提供有关分形几何的应用函数。本节主要目的是让读者了解一些分形几何的初步知识，以及一些简单分形图像的 MATLAB 产生方法。感兴趣的读者可以深入学习分形几何及其相关理论，也可以自己编写 MATLAB 函数，添加到 MATLAB 工具箱中进行使用。

11.2.1 分形几何基础

1. 分形几何的概念

曼德勃罗（Mandelbrot）于 1975 年创建的分形几何学（Fractal Geometry）不仅描述了存在于自然界中那些似乎很复杂的形态和模式，而且还提供了研究这些形态和模式的数学模型。计算机图形学与图像处理技术为分形学提供了研究分形物体的工具和手段，同时，分形学在用计算机显示、建模或模拟自然景物及各种奇异形状中起着重要的作用。

从表 11-1 可知，分形几何学与我们熟悉的欧几里德几何学相差甚远。海岸线的长度、云彩的体积、山地的面积等，这些量严格地说是无法测量的。以测量海岸线的长度为例，用长度为 r 的尺来度量海岸线，共得到 $N(r)$次测量次数，则海岸线长度为 $L=r\cdot N(r)$。按传统的观念，当 $r\to 0$ 时，$L\to$有限值，这个极限值就是海岸线的长度。而按 Fractal 观点，$L\to\infty$，即海岸线的长度不存在，因为海岸无论从多小尺度上来看，均不是直的，也就是说是处处不可微的，故长度不是描写海岸线的度量。海岸线无特征尺寸，必须从其他角度描写海岸线这类的自然对象。从传统的观念看，任何曲线从局部上看均是直线，这正是进行数学分析的基础，而欧几里德几何无法描述上述自然景象，只能描述如圆之类有特征尺寸的对象。

表 11-1 分形几何学与欧几里德几何学对照表

	分形几何	欧几里德几何
发展历史	20 多年	2000 多年
描述基础	无特征尺寸	对象的特征尺寸
适合对象	自然形状	人造
描述方式	过程	公式

以下是一个典型的分形例子，画出了一个漂亮的蝴蝶。

```
%分形程序
t=1:10000;
p=0.2*sin(3.*t)+sin(4.*t)+2*sin(5.*t)+1.9*sin(7.*t)-0.2*sin(9.*t)+sin(11.*t);
polar(t,p);
```

运行得到的结果如图 11-15 所示。

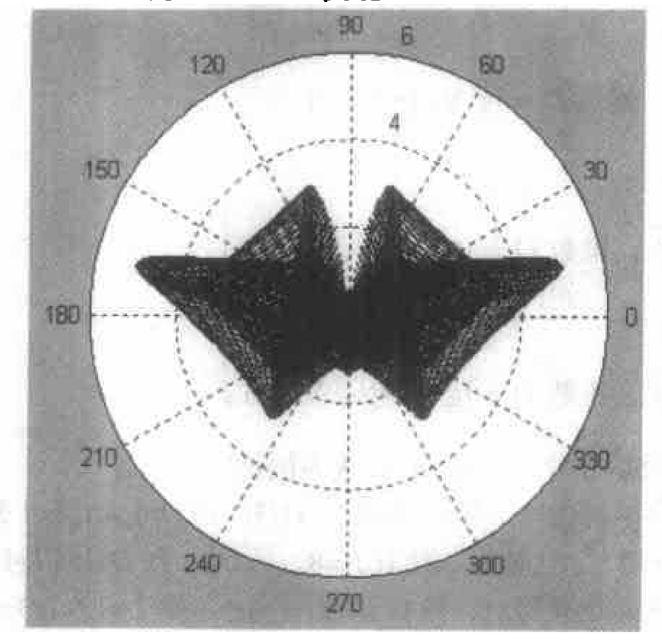


图 11-15　分形得到的蝴蝶

2．分形的特征——自相似性和分形维

自相似性是分形中的中心概念之一，它与维数的概念是密切相关的。分形几何描述的对象是自相似的（或统计意义上的相似）。自相似性用分形维数来描述。

若一自相似物由 N 部分组成，每一部分的尺度均是整体的 r（$r<1$）倍，则分形维数为

$$D = \lg N / \lg(1/r)$$

传统的 1、2、3 维是分形维式的特例，如图 11-16 所示。

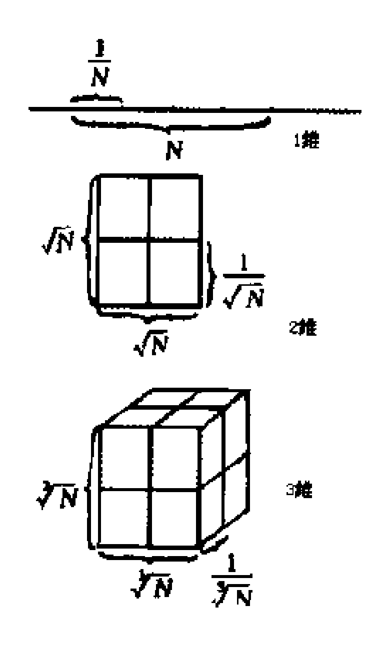


图 11-16　1,2,3 维的分形维数示意说明

对于 1 维来说：

$$r = 1/N\text{，维数 }D = \lg N/\lg N = 1\text{。}$$

对于 2 维来说：

$$r = 1/\sqrt{N}\text{，维数 }D = \lg N/\lg\sqrt{N} = 2\text{。}$$

对于 3 维来说：

$$r = 1/\sqrt[3]{N}\text{，维数 }D = \lg N/\lg\sqrt[3]{N} = 3\text{。}$$

然而分形维一般是分数，以 Koch 曲线为例说明如下。

将用替代时，便有 N=4、r=1/3，则 D=lg4/lg3=1.26。

而如果将用代替时，便有 N=8、r=1/4，则 D=lg8/lg4=1.5。

而如果将用代替时，便有 N=9、r=1/3，则 D=lg9/lg3=2。

由此可见：当 $1<D<2$ 时，Koch 曲线比简单的线条占据的空间要多，但比一个面占据的空间要少；当 $D\to 2$ 时，Koch 曲线变得更加弯曲，有更多的细节，故分形维 D 提供了一个曲线弯曲度的定量描述。尽管 Koch 曲线的分形维在 1～2 之间，但从拓扑学讲，它们是 1 维的，因为从曲线上移去一个节点，则曲线分成两段。为此，有人将传统维称为拓扑维。

3．统计自相似性

自然界中的物体不是严格自相似的，即在统计意义上是自相似的，但在不同尺度上的细节是有差别的。分形维对这类物体同样适用，以海岸线的长度为例。

$$L=r\cdot N(r)$$

从 Koch 曲线可知，$N(r)$ 从平均意义上看，与 $1/r^D$ 成正比，即 $N\times r^D=1$，故

$$L\propto r\times 1/r^D = 1/r^{D-1}$$

当 $D>1$ 时，若 r 减小，则 L 增大。从实验中得知，此时分形维的最佳值是：$D\in[1.15,1.25]$。

4．分形的实际意义

大量的实验表明，人对于那些分形维数（D）高于其拓扑维数（E）0.2 左右的计算机产生的景物感到最自然。实际上，海岸线的分形维是 D=1.2（对应的 E=1），地貌的 D=2.2（对应的 E =2），云彩的 D =3.3（对应的 E =3）。也就是说，人眼在自然环境下已经适应于看 $D-E$=0.2 左右的景物。

5．分形的分类

通过自身的比例变换和旋转变换所得到的分形称为确定性分形，如 Koch 曲线。带有随机因素的分形称为随机性分形。



11.2.2 MATLAB 在分形几何中的应用举例

本小节将举两个例子，来说明 MATLAB 在分形几何中的应用。

1. Koch 曲线的 MATLAB 程序

Koch 曲线由瑞典数学家柯曲（H.Von.Koch）在 1904 年首次提出，这是一个规则分形的典型例子。它的生成方法是把一条直线段三等分，将中间一段用夹角为 60° 的两条等长折线来代替，得出第一个生成元。把第一个生成元的每一条直线段用生成元迭代，得到第二个生成元。经过这样无数次的迭代，就呈现出一条无穷多弯曲的 Koch 曲线。Koch 曲线虽然处处连续，但处处不光滑，处处不可微，这类曲线变化多端，不好理解，难以掌握。借助于计算机，我们可以模拟这类曲线的生成过程，绘出它的图形，为分形几何的理论研究提供最直观的形象。下面给出绘制 Koch 曲线的 MATLAB 函数（存为 Koch 文件）.

```
function koch(s,t,n)

if size(s)~=size(t)
  disp('输入参数的维数必须相等');
else
  for j=1:n
    x=[];
    y=[];
    for i=1:length(s)-1
      d1=s(i+1)-s(i);
      d2=t(i+1)-t(i);
      x1=s(i)+[0,d1/3,(s(i+1)+s(i))/2-sqrt(3)/6*d2-s(i),2/3*d1,d1];
      y1=t(i)+[0,d2/3,(t(i+1)+t(i))/2+sqrt(3)/6*d1-t(i),2/3*d2,d2];
      if i==1
        x=[x x1];
        y=[y y1];
      else
        x=[x x1(2:5)];
        y=[y y1(2:5)];
      end
    end
    s=x;
    t=y;
    plot(x,y);
    %title('n='+int2str(n));
    xlabel('x');
    ylabel('y');
  end
end
```

使用这个函数，我们可以得到 Koch 曲线的不同应用。如果输入的初始折线是正三角形，则可以得到雪花。其 MATLAB 子程序是：

```
subplot(2,2,1);
```

```
koch([0,1/2,1,0],[0,sqrt(3),0,0],1);
title('n=1');
subplot(2,2,2);
koch([0,1/2,1,0],[0,sqrt(3),0,0],2);
title('n=2');
subplot(2,2,3);
koch([0,1/2,1,0],[0,sqrt(3),0,0],3);
title('n=3');
subplot(2,2,4);
koch([0,1/2,1,0],[0,sqrt(3),0,0],4);
title('n=4');
```

运行这个程序，我们可以得到漂亮的雪花，如图 11-17 所示。

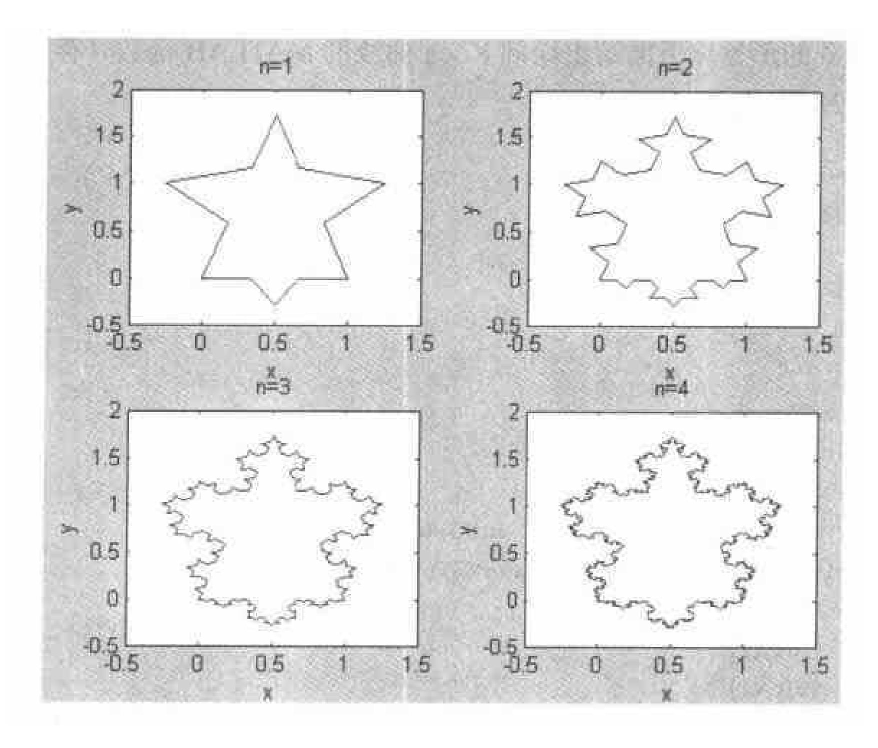


图 11-17　Koch 雪花图形绘制示意图

2. Sierpinski 垫片的 MATLAB 程序

Sierpinski 垫片的构造方法如下：取一个正三角形，将其四等分，舍去中间的一个小三角形，保留其余三个小三角形，得到第一个生成元；对第一个生成元的三个三角形分别按同样的方法操作取舍，即对第一个生成元的三个三角形用生成元迭代，得到第二个生成元；经过无数次迭代，就得出 Sierpinski 垫片。用计算机绘制 Sierpinski 垫片的 MATLAB 子程序如下（存为 Sierpinski.m 文件）。

```
function sierpinski(x,y,z,n)
%x 为初始三角形的第一个顶点的坐标
%y 为初始三角形的第二个顶点的坐标
%z 为初始三角形的第三个顶点的坐标
%n 为迭代次数
x1=x(1);
```

```
x2=x(2);
y1=y(1);
y2=y(2);
z1=z(1);
z2=z(2);
for j=1:n
  a1=[];
  a2=[];
  b1=[];
  b2=[];
  c1=[];
  c2=[];
  for i=1:length(x1)
    a1=[a1,x1(i),(x1(i)+y1(i))/2,(x1(i)+z1(i))/2];
    a2=[a2,x2(i),(x2(i)+y2(i))/2,(x2(i)+z2(i))/2];
    b1=[b1,(x1(i)+y1(i))/2,y1(i),(y1(i)+z1(i))/2];
    b2=[b2,(x2(i)+y2(i))/2,y2(i),(y2(i)+z2(i))/2];
    c1=[c1,(x1(i)+z1(i))/2,(y1(i)+z1(i))/2,z1(i)];
    c2=[c2,(x2(i)+z2(i))/2,(y2(i)+z2(i))/2,z2(i)];
  end
  x1=a1;
  x2=a2;
  y1=b1;
  y2=b2;
  z1=c1;
  z2=c2;
end
for i=1:length(x1)
  fill([x1(i),y1(i),z1(i),x1(i)],[x2(i),y2(i),z2(i),x2(i)],'b');
  hold on
end
hold off
```

运行以下 MATLAB 程序，我们可以得到 Sierpinski 垫片：

```
subplot(2,2,1);
sierpinski([0,0],[1/2,sqrt(3)/2],[1,0],1);
title('n=1');
subplot(2,2,2);
sierpinski([0,0],[1/2,sqrt(3)/2],[1,0],2);
title('n=2');
subplot(2,2,3);
sierpinski([0,0],[1/2,sqrt(3)/2],[1,0],3);
title('n=3');
subplot(2,2,4);
sierpinski([0,0],[1/2,sqrt(3)/2],[1,0],4);
title('n=4');
```

得到的结果如图 11-18 所示。

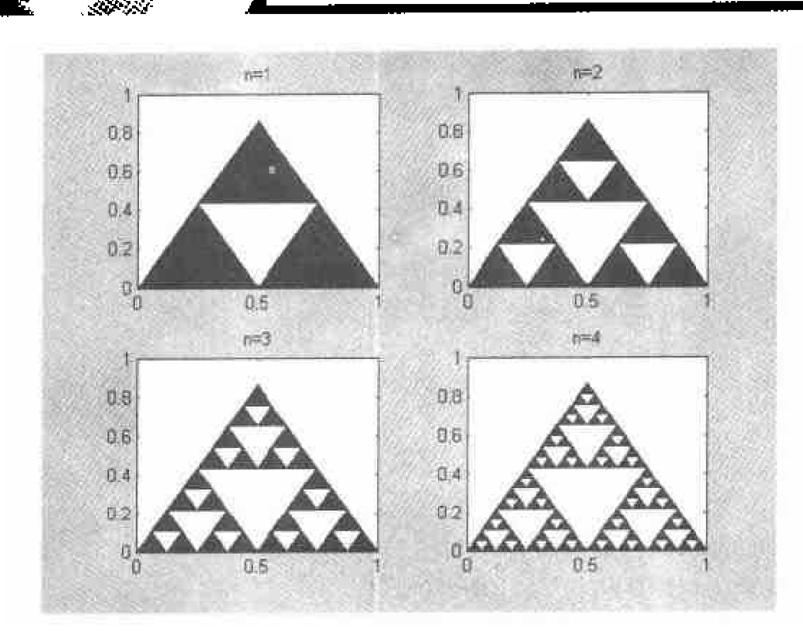


图 11-18 绘制 Sierpinski 垫片示意图

第 12 章　综合应用实例

对钢纹的区域标识在实际应用中，所需要解决的问题通常不是仅仅通过某一种图像处理方法就能得到比较理想的效果，而是需要综合应用多种方法，通过适当的组合才能达到图像处理的目的。因此本章将针对几个典型实例，结合使用前几章中介绍的图像处理方法，对处理的过程进行详细描述。

本章主要内容：

- 对不均匀亮度的校正
- 基于特征的逻辑

12.1　对不均匀亮度的校正

由于拍摄过程的原因造成图像中亮度不均匀的现象是很常见的，在这种情况下，再对图像进行边缘检测、模式识别等其他工作就会受到影响，因此要首先对不均匀亮度进行校正。

本例详细讲解了对一幅大米粒图像的不均匀亮度的校正。原始图像如图 12-1 所示，可以看出图像上中部的背景比下部的背景要亮得多。

```
I1=imread('rice.tif');
figure,imshow(I1)
```

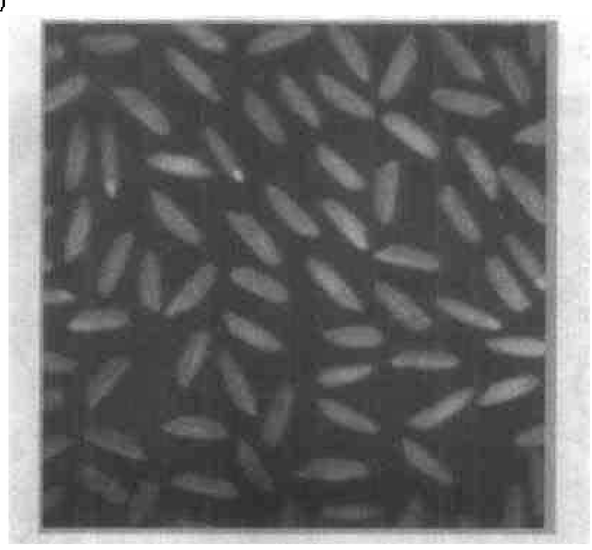


图 12-1　rice.tif 原始图像

（1）使用 blkproc 命令对背景的亮度进行粗略的估计，确定图像中每个 32×32 方块的最小值。下面画出了这个粗略估计的表面图，在 MATLAB 图形窗口中可以拖动坐标轴使图形旋转。实现代码如下：

```
I2=im2double(I1);
bg32=blkproc(I2,[32 32],'min(x(:))');
surf(bg32),rotate3d on
```

图形窗口如图 12-2 所示。

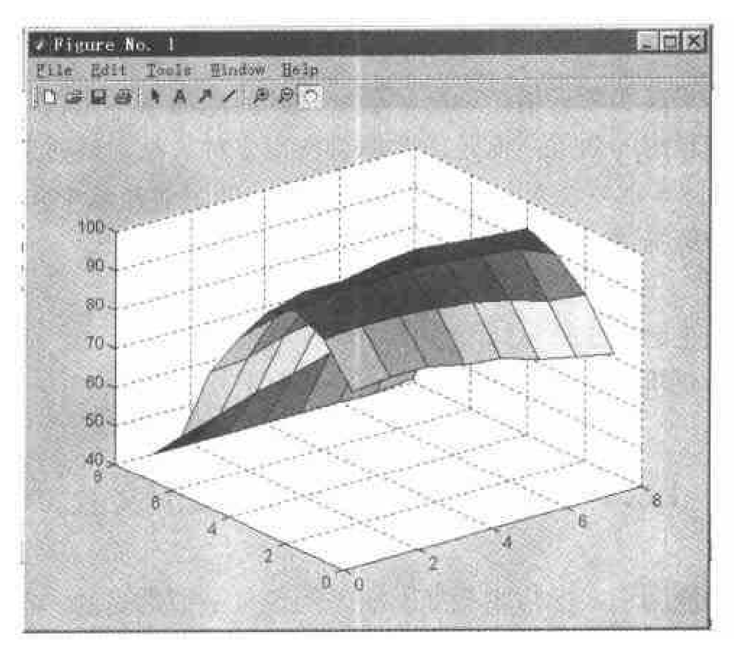


图 12-2　对背景亮度的粗略估计的表面图

（2）使用 imresize 函数将粗略估计扩展到与原始图像一样大小。使用三次插值对数据进行平滑。代码如下：

```
bg256=imresize(bg32,[256 256],'bicubic');
figure,imshow(bg256)
```

处理的效果如图 12-3 所示。

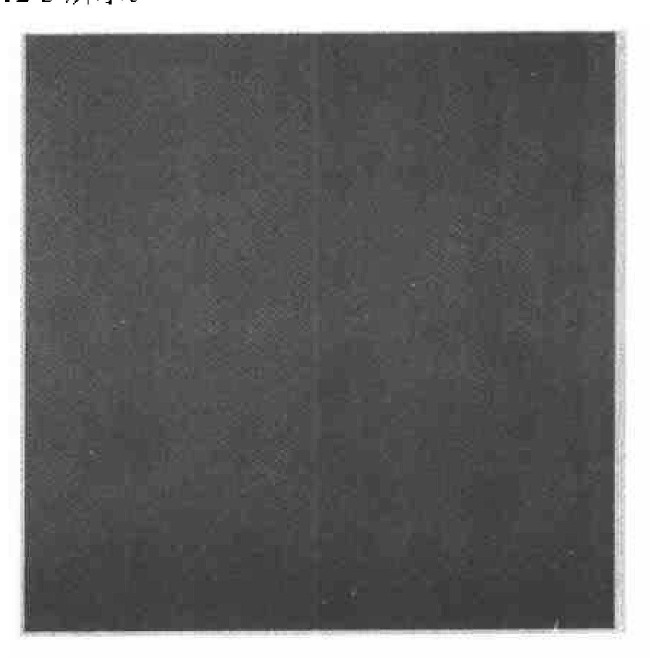


图 12-3　imresize 函数处理结果

（3）从原始图像中减去前面计算出来的亮度，校正不均匀性，但此步操作同时会使大米粒也变暗。代码如下：

```
d=I2-bg256;
figure,imshow(d)
```

处理的效果如图 12-4 所示。

（4）将像素的灰度值调整到整个灰度级，使得大米粒变得较亮，具有更丰富的细节。代码如下：

```
adjust=imadjust(d,[0 max(d(:))],[0 1],1);
figure,imshow(adjust)
```

处理的效果如图 12-5 所示。

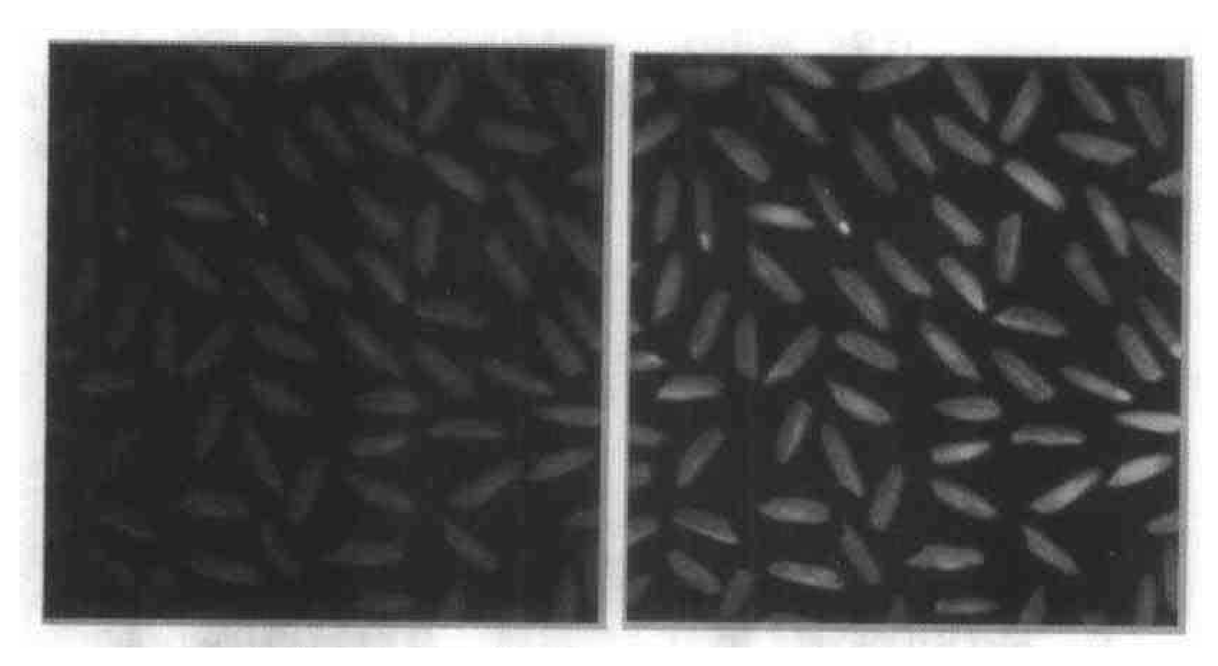


图 12-4　用图 12-3 减去原始图像的结果　　　图 12-5　灰度调整后的最终处理结果

将图 12-5 与原始图像进行比较，很明显可以看出处理后的结果中背景亮度分布均匀，米粒与背景之间的对比明显，米粒清晰易于辨识处理。

12.2　基于特征的逻辑

下面分别以合成二值图像和细菌的显微图像为例，实现基于特征的逻辑。

1．将基于特征的逻辑用于合成二值图像

在图 12-6 和图 12-7 中，找出 dots 图像的特征和 box 图像的特征相交叠的部分。代码如下：

```
load imdemos dots box
imshow(dots);
figure,imshow(box);
```

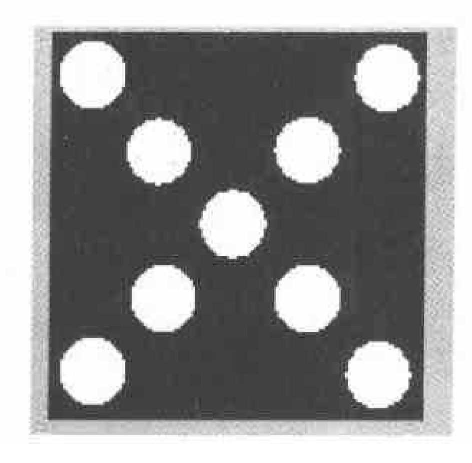


图 12-6　dots 图

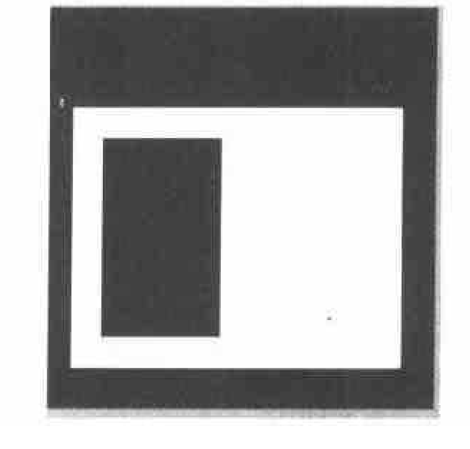


图 12-7　box 图

（1）对两幅图像进行逻辑与操作，辨识在两幅图像中值均为 on 的像素，代码如下：

```
logical_and=box&dots;
figure,imshow(logical_and);
```

处理后的图像如图 12-8 所示。

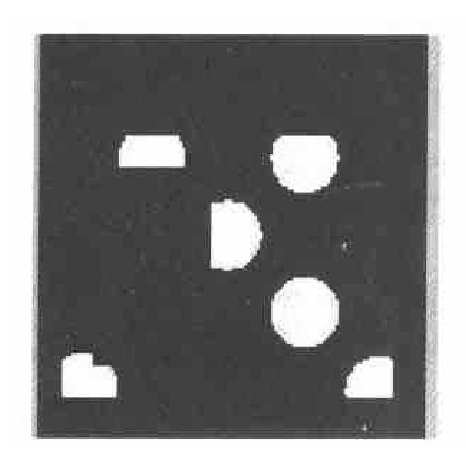


图 12-8　两幅图逻辑与的结果

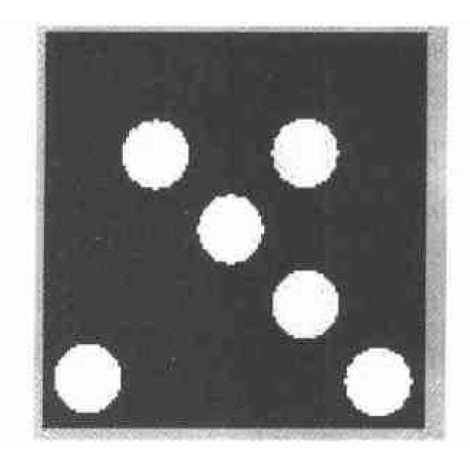


图 12-9　基于特征的与的结果

（2）将前一步逻辑与中值为 on 的像素作为 bwselect 函数的输入，实现基于特征的与，代码如下：

```
[r,c]=find(logical_and);
feature_and=bwselect(dots,c,r);
figure,imshow(feature_and)
```

处理后的图像如图 12-9 所示。

2．将基于特征的逻辑用于细菌的显微图像

在图 12-10 中，我们要辨识出哪些细菌包含一个或多个亮颗粒，确定包含颗粒的细菌的个数，而不是确定颗粒的个数，因为有一些细菌可能包含多个颗粒。

```
load imdemos bacteria
figure,imshow(bacteria)
```

（1）对图像取阈值 100，将细菌进行分割，得到二值分割图像，代码如下：

```
bact_bw=~(bacteria>100);
figure,imshow(bact_bw)
```

处理的结果如图 12-11 所示。

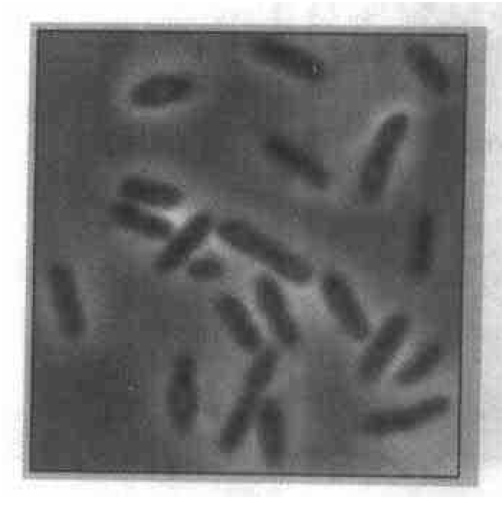


图 12-10 bacteria 原始图像

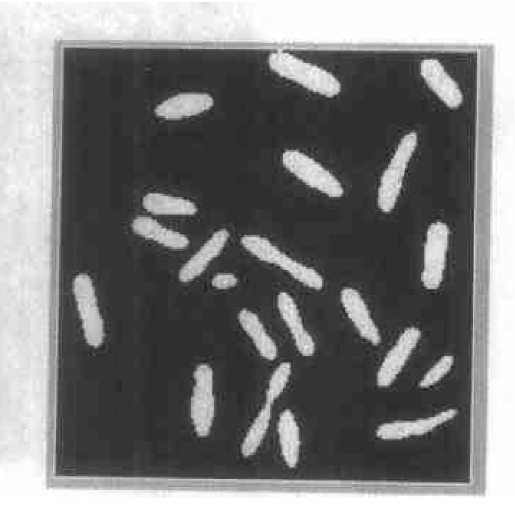


图 12-11 得到的二值分割图像

（2）对原始图像应用拉普拉斯滤波器，对滤波结果取阈值-4，然后与上一步中得到的二值分割图像进行逻辑结合。得到的图像中显示了细菌细胞中的颗粒。代码如下：

```
filtered=filter2(fspecial('laplacian'),bacteria);
filtered_bw=(filtered>-4);
bact_granules=filtered_bw&bact_bw;
figure,imshow(filtered,[])
figure,imshow(bact_granules)
```

处理的结果如图 12-12 和图 12-13 所示。

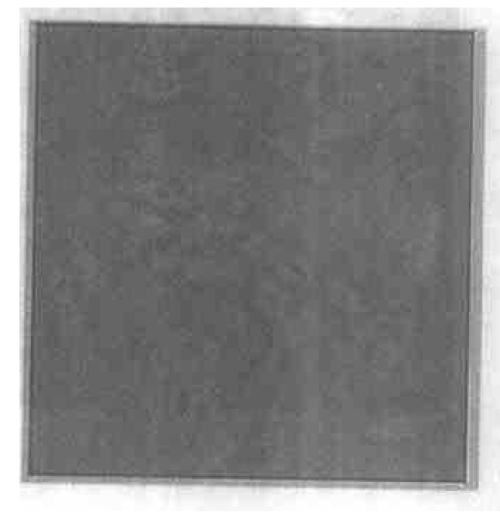


图 12-12 应用拉普拉斯滤波器的结果

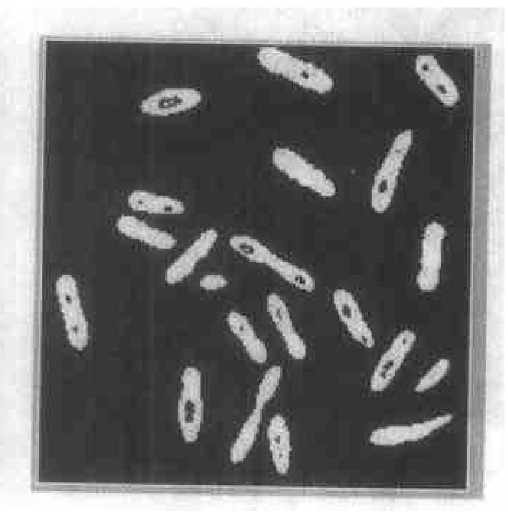


图 12-13 逻辑结合的结果

（3）腐蚀第一步中得到的分割图像，在结果中选择 bact_granules 为 0 的区域，从而将颗粒分离开来。腐蚀后可以避免检测到细菌细胞边缘上的点。代码如下：

```
granules=erode(bact_bw);
granules=granules&(bact_granules==0);
figure,imshow(granules)
```

处理的结果如图 12-14 所示。

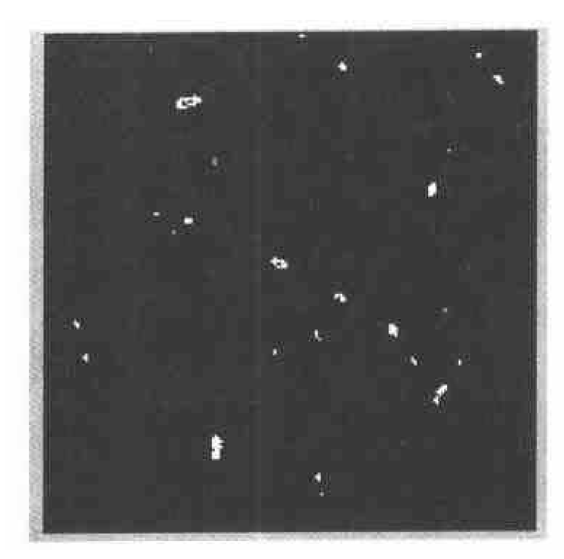


图 12-14　检测到的亮颗粒

（4）辨识取过阈值的简单图像中有颗粒的对象。代码如下：

```
[r,c]=find(granules);
result=bwselect(bact_bw,c,r);
figure,imshow(bacteria)
figure,imshow(result)
```

原始图像和处理的结果分别如图 12-15 和图 12-16 所示。

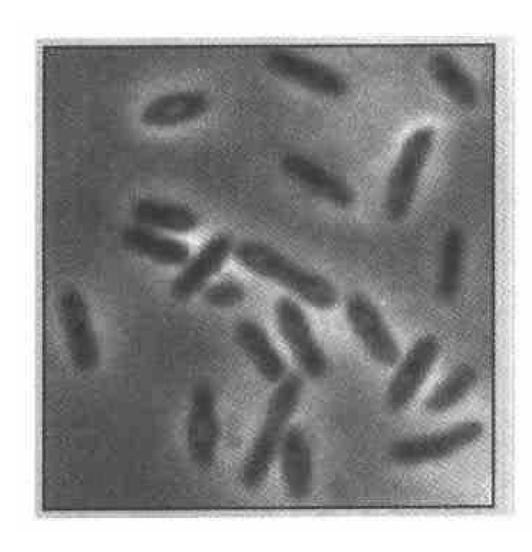


图 12-15　bacteria 原始图像

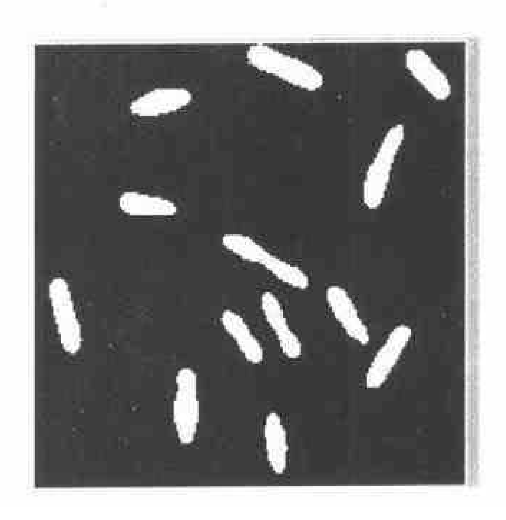


图 12-16　辨识出的有颗粒的细菌细胞

12.3　对钢纹的区域标识

本例应用了取两次阈值、基于特征的逻辑、二值形态学和相连成分的标识，确定了钢的显微图像（见图 12-17）中颗粒的边界，标识了不同的颗粒。

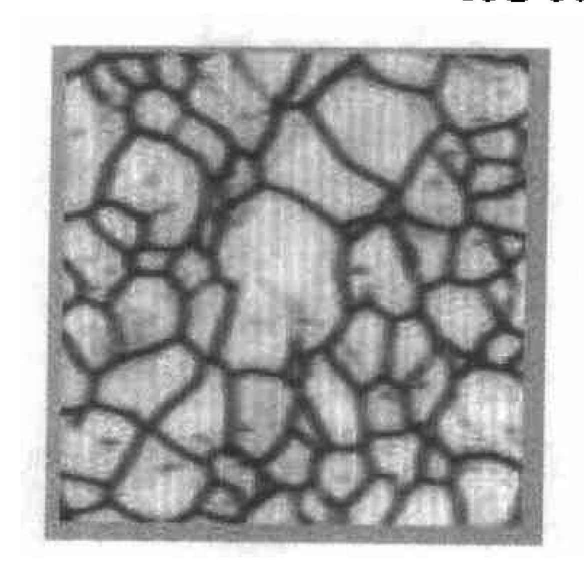


图 12-17　钢的原始图像

（1）对钢的图像取两个不同的阈值 70 和 210，代码如下：

```
load imdemos steel;
figure,imshow(steel)
bw_70=steel>70;
bw_210=steel>210;
figure,imshow(bw_70)
figure,imshow(bw_210)
```

处理的结果如图 12-18 和图 12-19 所示。

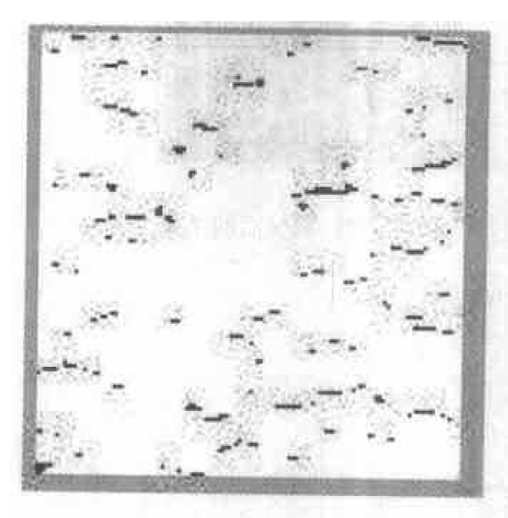


图 12-18　取阈值 70 的图像

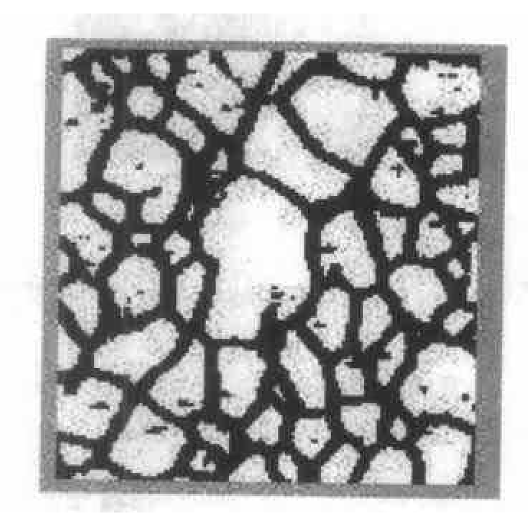


图 12-19　取阈值 210 的图像

（2）用图 12-18 中的黑色点来选取图 12-19 的求反图像中的白色对象，这样可以有效地去除图 12-19 中的较小区域。代码如下：

```
[r,c]=find(bw_70==0);
bw_clean=bwselect(~bw_210,c,r,8);
figure,imshow(bw_clean)
```

处理的结果如图 12-20 所示。



图 12-20 去除了图 12-19 中的较小区域的结果

（3）将去除了较小区域的结果骨架化，然后从骨架化的图像中剪去尖刺像素。代码如下：

```
bw_skel=bwmorph(bw_clean,'skel',6);
figure,imshow(bw_skel)
bw_pruned=bwmorph(bw_skel,'spur',8);
figure,imshow(bw_pruned)
```

处理的结果如图 12-21 和图 12-22 所示。

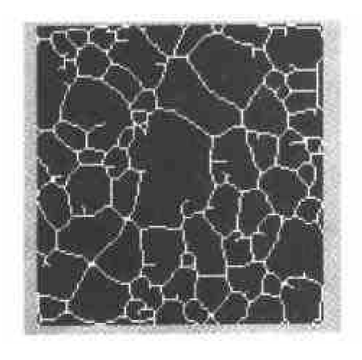


图 12-21 骨架化的结果

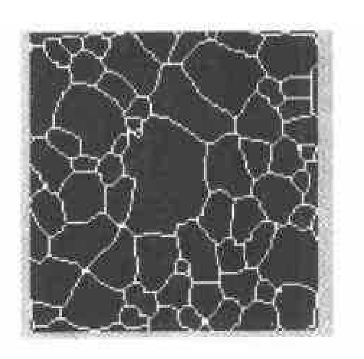


图 12-22 剪枝后的结果

（4）颗粒的边界图就是剪枝图像的逻辑非。代码如下：

```
grain_boundaries=~bw_pruned;
figure,imshow(grain_boundaries)
```

处理的结果如图 12-23 所示。

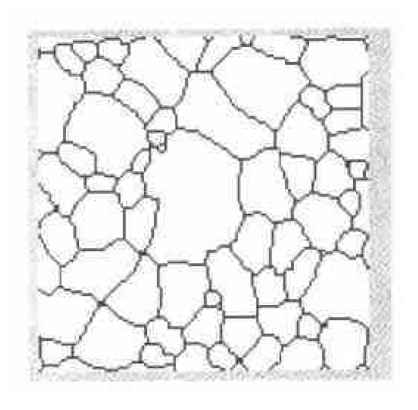


图 12-23 剪枝图像的逻辑非

（5）将对象映射成不同的颜色显示出来。Bwlabel 命令可以将二值结果中的不同对象标识出来。代码如下：

```
[labeled,N]=bwlabel(grain_boundaries,4);
colored=ind2rgb(labeled+1,[0 0 0;jet(N)]);
figure,imshow(colored)
```

处理的结果如图 12-24 所示。

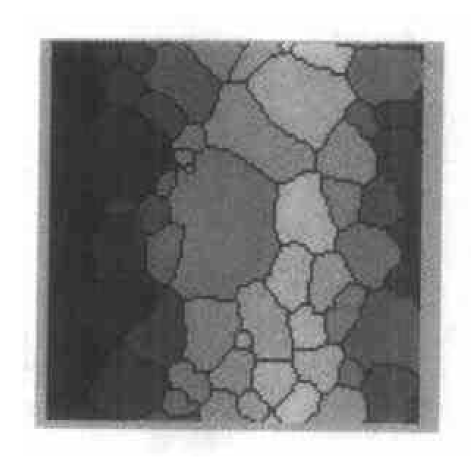
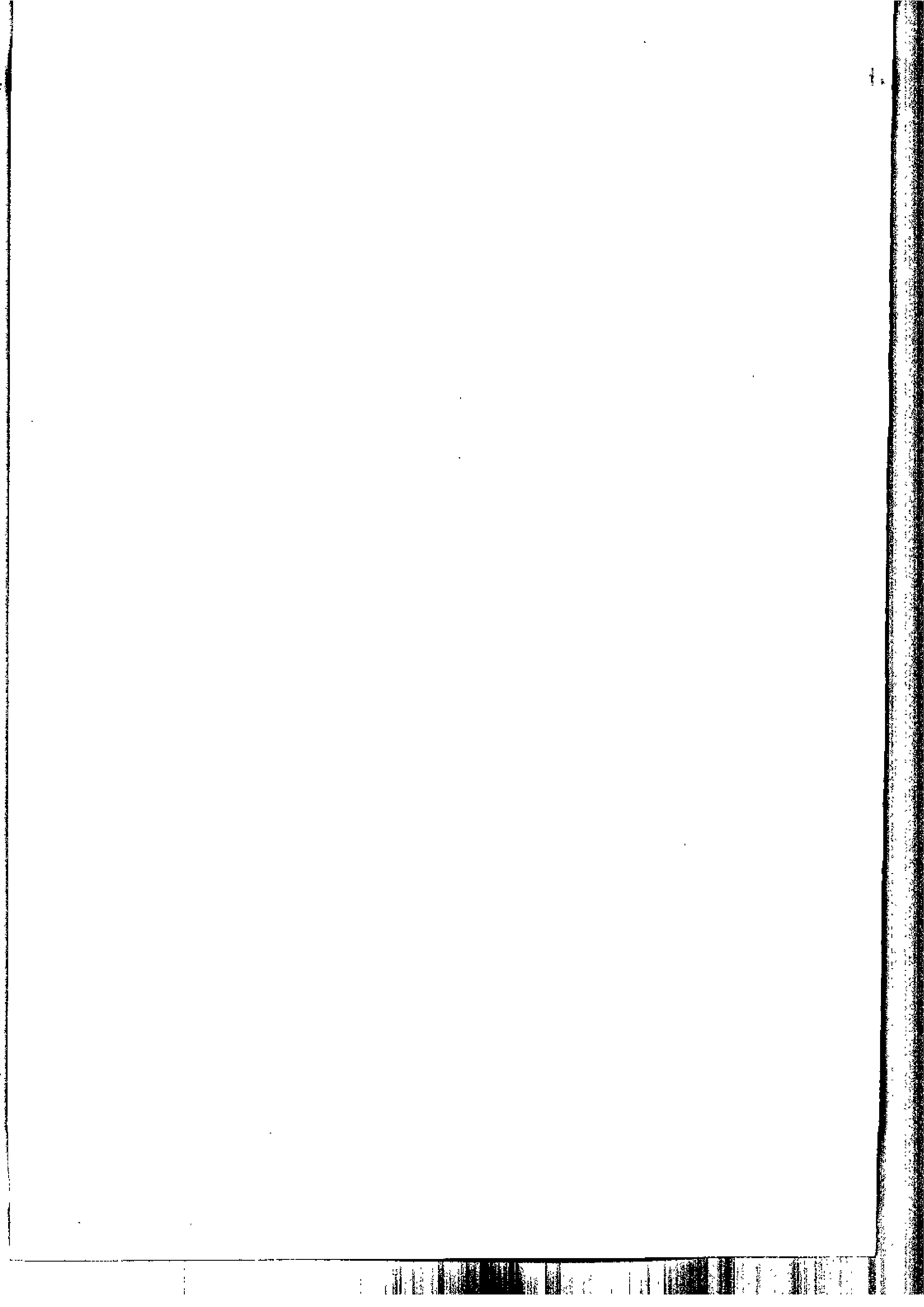


图 12-24 伪彩色图像



第 13 章　图形用户界面设计

图形用户界面（GUI）在 MATLAB 程序开发中占着举足轻重的作用。一个好的界面不仅有利于用户快速地掌握程序的操作流程，有效地使用程序，同时它也有利于开发者展示 MATLAB 平台下开发的技术。GUI 在科研实践和工程实践中有着广泛的应用，尤其是在图形处理技术、人工智能技术等方面。用户也许对此深有体会。在 MATLAB 的 DEMO 中，每一个实例都是 GUI 成功应用的例子。一个个栩栩如生的 DEMO 实例逐步引导用户了解、熟悉 MATLAB 的使用，读者是否也萌发了同样的念头，为自己的程序也创建一个 GUI？那么，在 MATLAB 下怎么创建 GUI？什么样的 GUI 才是比较成功的？本章希望能够给读者一个满意的答复。

本章主要内容：

- GUIDE 开发环境介绍
- GUI 设计
- GUI 实现

13.1　引　言

在科研实践中，越来越多的人放弃 VC，选用 MATLAB 来开发界面。这种现象的出现存在着以下两方面的原因。首先，开发者的重点是放在核心算法的设计，GUI 只是为了向客户演示技术的方便，相比较而言，VC 太复杂，需要开发者做大量的底层工作。其次，随着 MATLAB 在工程应用中的推广，许多核心的算法是用 MATLAB 实现的，因此，选用 MATLAB 的界面可以做到代码的无缝连接，不需涉及到 MATLAB 复杂的接口技术。

本章的内容大致是这样安排的。首先，介绍一个简单实例。通过该实例，读者会对 MATLAB 下的用户界面开发过程有一个大致的了解，同时初步掌握 MATLAB 图形用户界面开发环境 GUIDE 的使用。一个成功的 GUI 必须建立在精巧的设计之上。因此，本章接下来侧重分析 GUI 的设计过程中需注意的一些问题，也可以说，开发者必须遵循的一些设计原则。只要遵循这些原则，相信你所开发出的界面会很容易被用户所接受。本章最后用了大量篇幅结合实例对 GUI 实现中的技术细节做了介绍。开发者如果掌握了这些细节，就可以在用户界面上自由发挥，使图形界面具有更多的个性。

开始本章之前，读者需要对图形句柄的概念有大致了解。如果你有用 VC、VB 开发界面的经验的话，当然更好。因为 MATLAB 下的界面开发也使用了当今的主流图形开发技术，开发环境大同小异，很多地方有相似之处。你在使用中往往会触类旁通。如果你是一个新手的话，也不要害怕。本章独立成章，深入浅出的介绍会让你在读完本章之后，晋级到高手的行列。

13.2　GUIDE 开发环境介绍

既然能够用 MATLAB 像在纸上一样求解数学问题，为什么不能像在纸上一样，用 MATLAB 把图形用户界面画出来？当然可以。解决这个问题的方法便是使用 MATLAB 中的 GUIDE。

GUIDE 实际上是一套 MATLAB 工具集，由五个部分组成：属性编辑器，控件面板，回调函数编辑器，调整工具，菜单编辑器。有了这些工具，我们可以在短短几分钟内设计出一个漂亮的 GUI。这只是问题的第一步。一旦有了界面，后面的工作便是通过 GUIDE 的回调函数编辑器编写响应函数代码，来响应用户的界面操作。下面的例子便是将一条函数曲线显示在 GUI 中，然后用 GUIDE 向这个 GUI 中添加按钮，编写按钮的回调函数。于是，通过按钮可以控制函数曲线的显示过程。这当然只是一个简单的用户交互界面，但是通过它，你可以明白什么是 GUIDE，如何使用 GUIDE。

例程 13-1 绘制函数曲线 t×sin(t)。

例程 13-1

```
t = 0:0.1:20;
%绘制函数曲线，生成一个图形 Figure
plot(t,t.×sin(t))
get(gcf, 'Color')
%gcf 函数返回当前 Figure 的句柄，get 得到当前 Figure 的颜色属性
ans =0.8000 0.8000 0.8000
%MATLAB 中的所有颜色以 RGB 的形式存储，[1,1,1]是白色，[0,0,0]是黑色
%此时，可以用命令行的形式激活属性编辑器
propedit(gcf)
```

输出结果如图 13-1 所示。

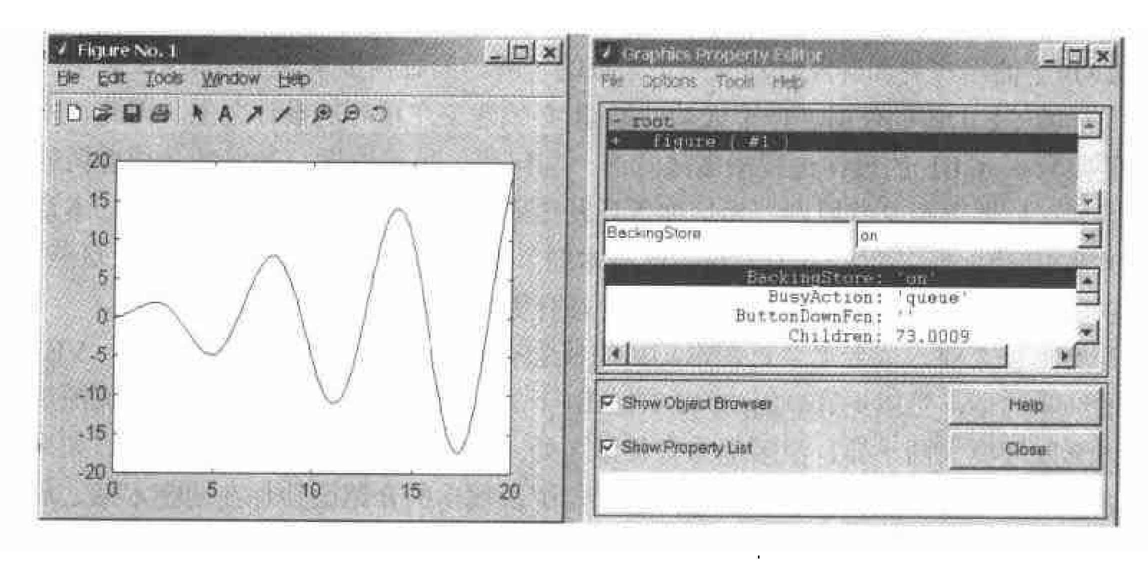


图 13-1　命令行产生的图形（左图）与激活的属性编辑器（右图）



在属性编辑器的【Options】菜单中，单击【Show Object Browser】命令。从弹出的对象浏览框中，可以发现 Figure 所包含的所有对象，这些对象以树形结构排列。将这个对象树全部展开后，你将会发现图形 Figure 中出现的任何一个可视部件都能够在这个对象树中找到它所对应的对象。选中一个对象，下面的属性列中将会出现该对象所具有的全部属性，以及属性允许的取值。比如图形 Figure 的“color”属性，允许的取值是颜色表中的任何一种颜色。

13.2.1 从控制面板添加按钮

在属性编辑器的【Tools】菜单中单击【Control Panel】命令，将会弹出控制面板的界面。

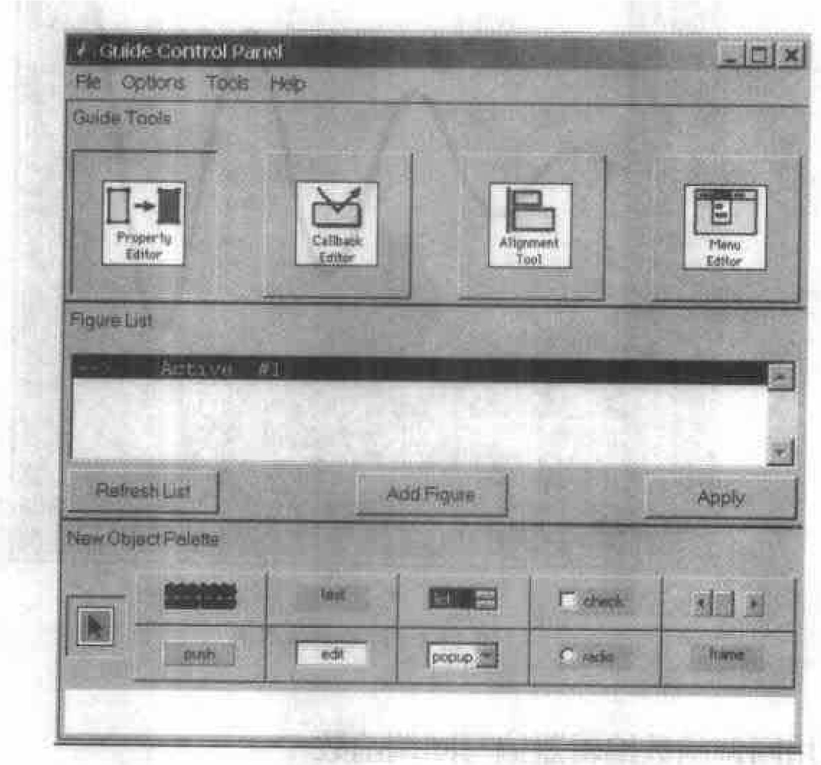


图 13-2 控制面板

该面板分为三个部分：

- 上面一部分是 GUIDE 工具列表。
- 中间一部分是 GUIDE 所控制的图形（Figure）列表。
- 最下面一部分是可以向 Figure 中添加的对象，包括按钮、文本框、复选框、列表框等。

从该控制面板的界面，可以发现【Property Editor】按钮处于“Push in”状态。这是因为属性编辑器已被打开，并被摆在桌面上。

中间的图形列表可以显示各个图形（当前只有一个图形）的状态。当该图形处于“Controlled”状态时，用户可以随意地移动被控图形上的对象，比如按钮、坐标轴等。GUIDE 会自动计算这些对象移动后所处的位置，并把位置坐标的数值放入 MATLAB 的代码中。

当设计者将图形上的一切都摆放到位后，可以把图形的状态切换至“Activated”。这时，对图形中对象的拖拉将无效。

在控制面板下部单击待添加的对象，这里是“button”。当鼠标移动到被控图形区域时，鼠标箭头将变成一个带加号的箭头。此时，便可通过拖拉鼠标改变按钮的大小，直到符合你的需要。按照以上步骤，便可顺利地将一个按钮添加到图形界面上。这时，该按钮仍处于被选中状态。接下来，还需要在属性编辑器中修改按钮的“string”属性为“grid on”（注意，不要漏掉双引号），出现如图 13-3 所示的界面。

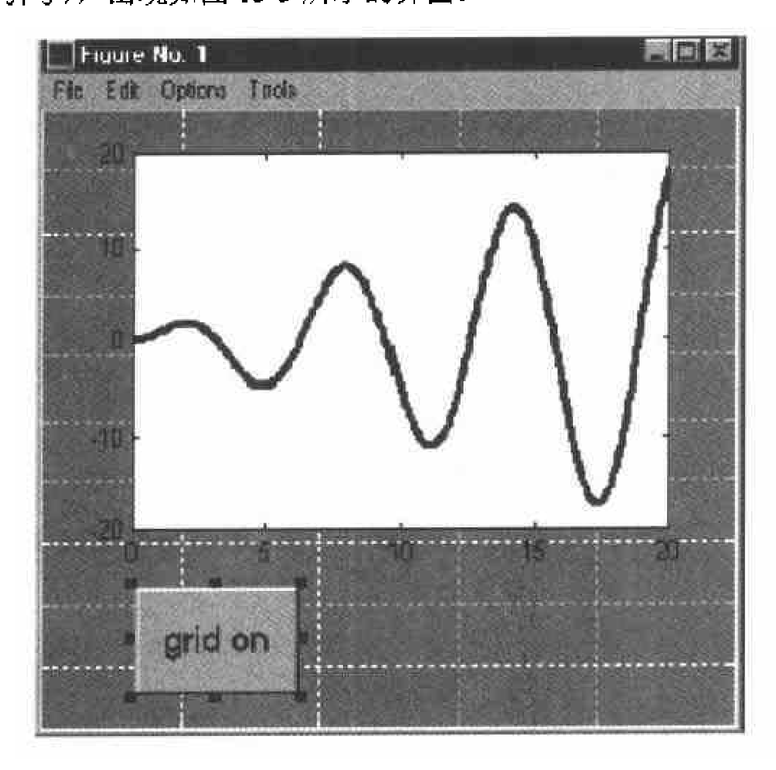


图 13-3　函数曲线界面

13.2.2　使用回调函数编辑器编写回调函数

目前，我们只是把一个按钮放到了界面上，还没有定义按下该按钮后应执行的动作。下面使用回调函数编辑器来完成这一任务。首先，在按钮对象被选中的情况下，激活回调函数编辑器。方法是在控制面板的菜单【Tools】中单击【Callback Editor】命令。然后，在下拉菜单中单击“Callback”，在编辑框中输入“grid on”。

重复以上步骤，将“grid off”按钮加入界面。

13.2.3　激活图形

最后一步便是将图形从编辑状态切换到激活状态，使前几步的操作生效。具体方法是回到控制面板下，在图形列表中选中被控图形，单击该图形后，面板上的【Apply】按钮有效，再单击一下【Apply】按钮，图形便处于激活状态了。这时它便是一个简单的 GUI

界面。

13.3 GUI 设计

设计包括编写源代码之前的所有工作。好的开始是成功的一半。设计者必须重视设计工作，不要急于编写源代码，否则，编程员思路不清晰，开发过程中会走很多弯路，造成开发效率低下，并且设计出的 GUI 也不一定让人满意。那么如何进行成功的设计，如何避免上述可能出现的问题？本节将从两个角度予以探讨：一方面论述了设计 GUI 的指导原则，另一方面介绍了设计过程中应遵循的步骤。以上两点已被许多 GUI 开发者的实践所证明，希望能够对你有所启发。

13.3.1 指导原则

什么才是优秀的设计？对此，设计专家已达成了共识，优秀设计的标准是普遍适用的，GUI 设计也不例外。也许这样讲有点方法论的味道，会让你觉得言过其实。但是亲爱的读者，注重这些基本原则，你在以后的设计过程中会大受裨益，尤其在这里，按照规律办事永远是一条真理。

可以用 9 个字来概括优秀设计的标准：简洁性，一致性，熟悉性。

- 简洁性，意味着界面简单明了，直接清晰，用户可以很快提取出对自己有用的信息。
- 一致性，意味着设计系统的各个环节应遵从统一的、简单的规则，保证不出现例外情况。
- 熟悉性，意味着系统设计时尽量保持与该类流行界面的相似，比如 VC 的对话框风格，视图的风格，MATLAB 的 Figure 风格。这样用户操作起来就容易上手，误操作率比较低。

以上三点也许过于抽象，具体过程中不易操作。其实，只要在设计整个过程中贯穿以人为本的设计理念，设计出的界面自然会达到以上要求。设计者应该经常考虑这样两个问题：

- 用户在没有熟悉界面之前，完成一个操作需要多长时间？
- 用户在熟悉界面后，完成一个普通操作需要多长时间？

简洁性、一致性、熟悉性围绕着上述两个问题展开，它们有时存在着矛盾的地方，比如会为了一致性而牺牲简洁性。这种取舍的标准应着眼于用户，要看是否便于用户操作。用户应该是设计者心中的上帝。

1．简洁性

简洁是我们设计者所要达到的主要目标。简洁是一种美。通过 GUIDE，我们很容易向我们的界面上添加许多功能，但功能强大并不意味着一切，有时一些功能画蛇添足。本来 GUI 是从图像的角度帮助用户理解这个客观世界，但是缤纷芜杂的表面现象却掩盖了事物的真相，影响用户作出正确的判断，这是不可取的。设计者一定要警惕这种盲目追求功能

强大的心理。下面举出的一些例子，也许会给你带来一些启发。

- 注重形式：三维形体的观察

图 13-4 和图 13-5 对比显示了一个三维形体加网格和不加网格的视觉效果。通过对比，我们可以看出网格画蛇添足，遮挡了部分视线。

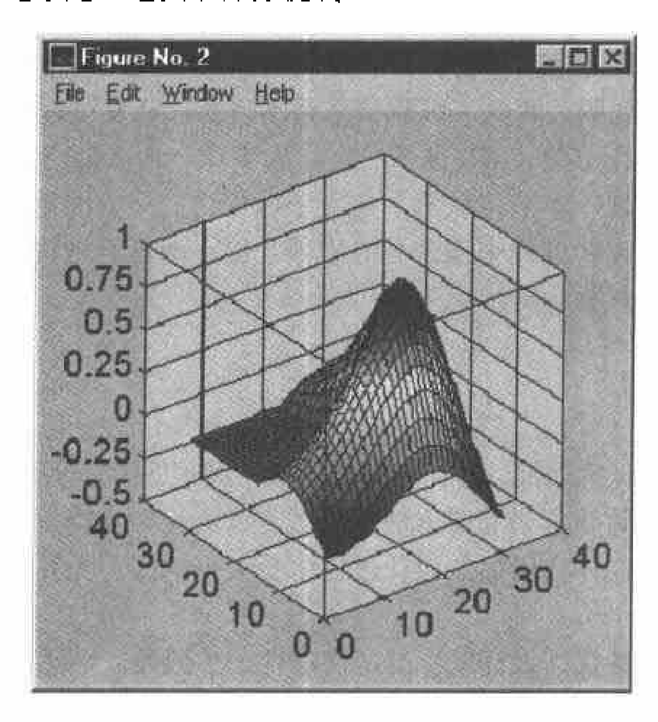


图 13-4 添加网格的视觉效果

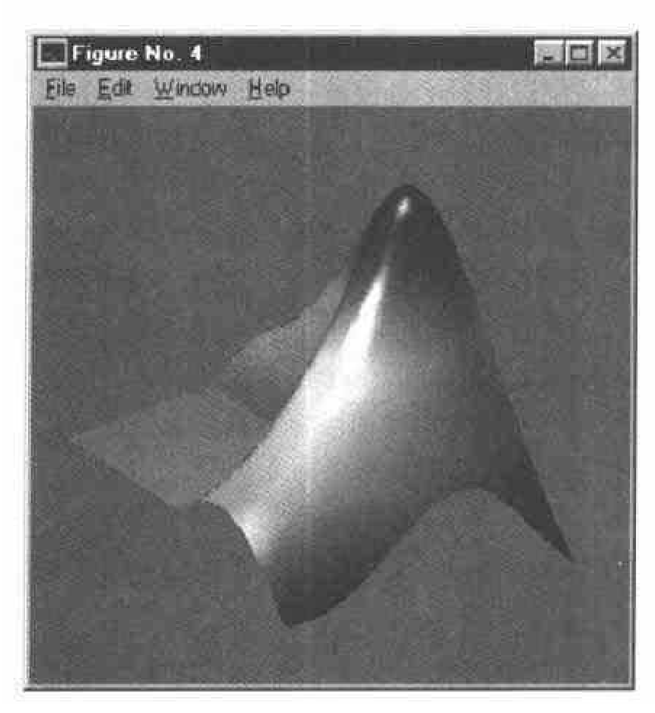


图 13-5 取消网格的视觉效果

图像往往给人一种直观的感受，告诉人们事物发展的趋势，数字有助于将这种趋势量

化，增强人们对发展趋势的认识。但是这并不意味着图像必须借助于数字。比较上面两个界面，你会发现左图上的网格并没有起到它应起的作用，相反还是个累赘。为什么不把它去掉？下边的图是多么优美，一眼望上去，你马上对这个三维图形的凸凹形成深刻的印象，立即做出判断——这个图形对应的函数含有一个全局极大点及该采用何种算法来求解这个极大点。

- 缩小交互区域

如果能在一个界面中完成的事情，绝对不要放在两个界面中去完成。这也是操作简洁性的一种间接体现。在 MATLAB 运行环境下，运行下列指令：

```
?bandemo
```

该演示程序是为了显示各种最优化算法对 banana 函数的处理效果。banana 函数是最优化理论中的一个著名问题。由于该曲线中间的曲率半径特别小，所以很多最优化算法不会收敛到极值点。因而用这个函数来检验算法的收敛性特别有效。

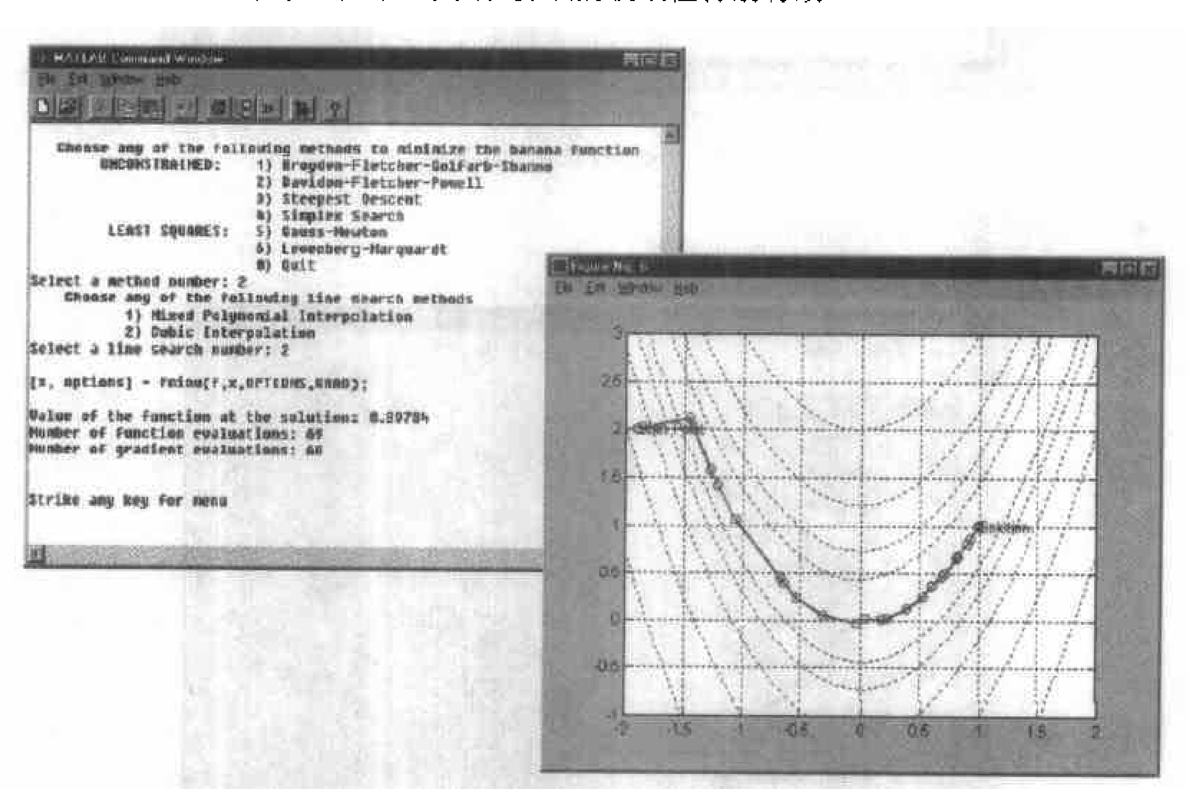


图 13-6　banana 优化算法

如图 13-6 所示，上述过程操作起来很麻烦，用户需要在一个窗口内完成各种算法的选择，在另一个窗口内观察算法的处理效果，这有悖于尽量缩小交互区域的原则。如果将这两个窗口集成在一起，使用起来的效果截然不同。在 MATLAB 环境下运行下列指令：

```
bandem
```

运行结果如图 13-7 所示。

通过按钮，用户可以直接在该界面上选择算法，无须在两个界面上进行切换。这种设计加深了用户对各种算法运行效果的直观理解，增强了图形的可读性。

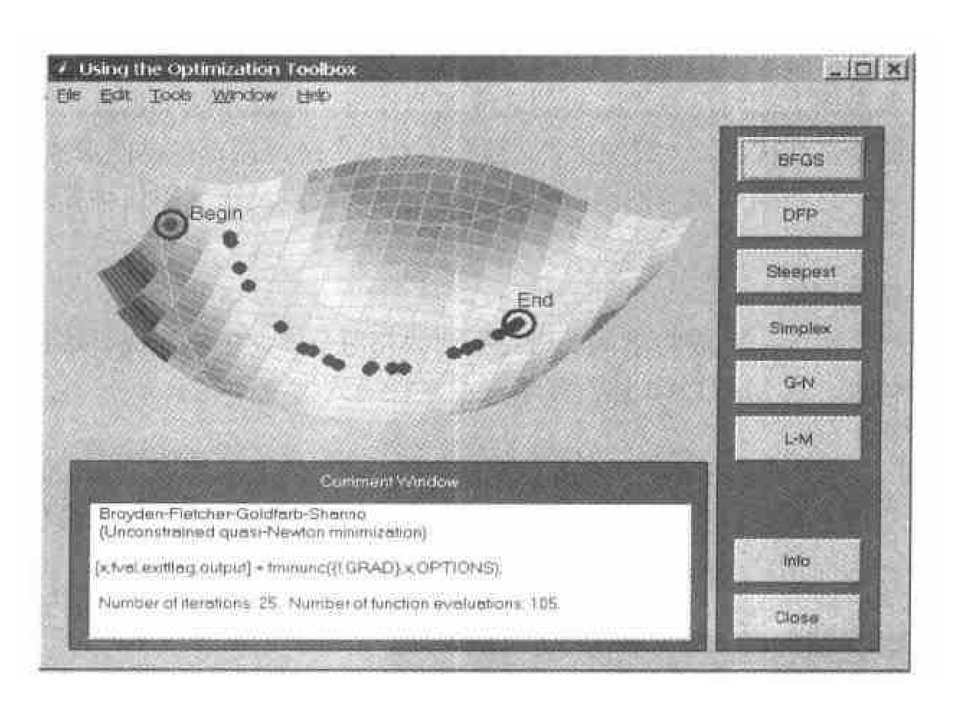


图 13-7　banana 优化算法界面

- 用图形输入代替数值输入

图 13-8 说明了三维形体观察视角的变换。

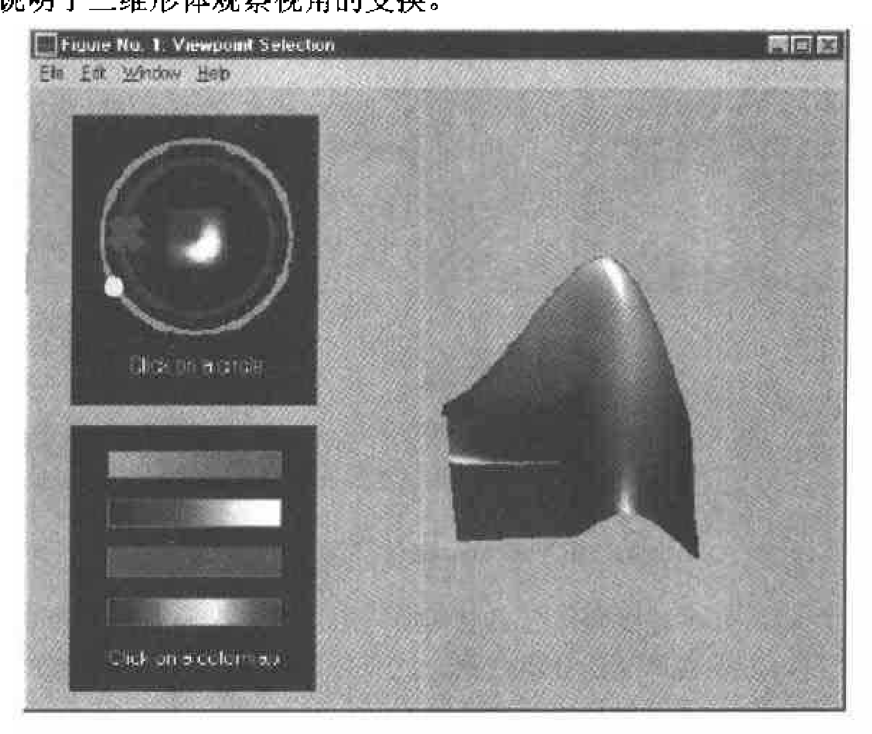


图 13-8　三维形体观察界面

图中用环绕几何形体的两个圆圈来表示镜头和光源的轨迹，用户可以通过拖拉“X”符号改变镜头的位置，拖拉圆点改变光源的位置。

对于用户来说，在图形界面下，图形输入比数值输入更加方便。为了从各个角度来观察一个三维几何形体，我们需要不断进行坐标系的变换。用数值输入的方法，困难程度难

想象，其复杂性会让用户望而却步，图形输入的方式则很好地解决了上述问题。

2. 一致性

一致性的含义很广泛，既包括操作指令的前后一致，也包括操作界面中各种图形对象摆放位置的一致性。这个原则基于以下认识：用户的操作经验应有助于用户完成后面的操作。设计者不应该给用户太多的意外，太多的意外会让用户一头雾水。前面执行这个命令是一种用法，后面却是另外一种用法，于是产生了歧义。优秀的界面设计往往注重这个问题，请看图 13-9 所示的例子。

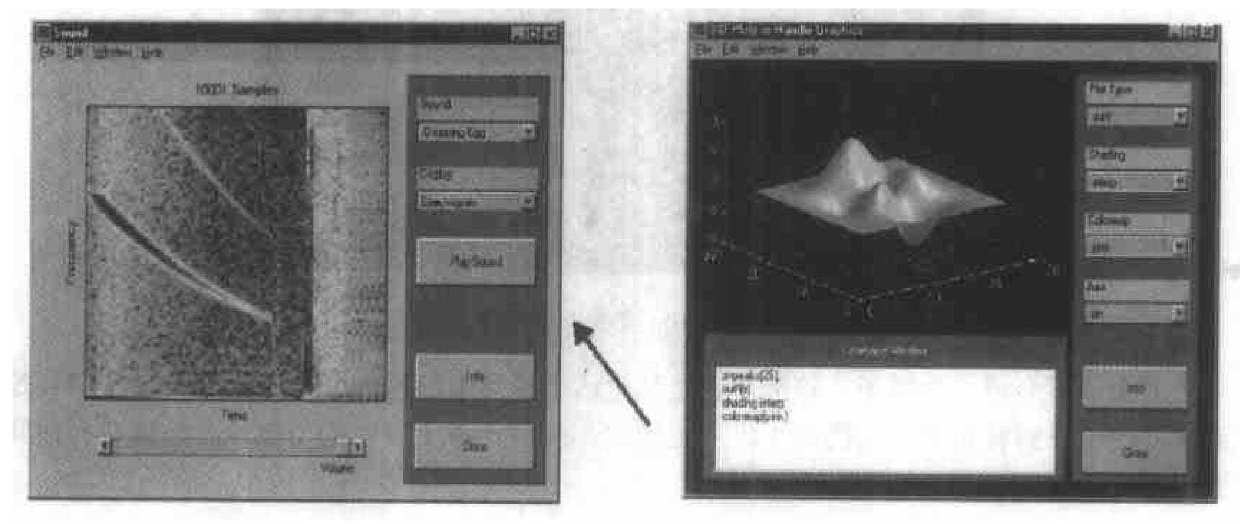


图 13-9 界面一致性比较

请注意，这两个界面中【Info】和【Close】两个按钮的摆放位置及其作用。虽然这两个界面是用来演示不同的技术，但是用户对其中任何一个界面的操作经历绝对有助于加深用户对另一个的了解。另外，还请读者注意界面中待处理图形和交互按钮的摆放位置。

也许有人对此不以为然，认为这些细节无关大局。但是作为设计者，必须处处站在用户的角度考虑问题。如果我是一个新手，我清楚我现在要干什么吗？我知道我下一步要干什么吗？这两个问题也许有助于加深开发者对以上细节的考虑。

3. 熟悉性

我们设计界面的目的是为了对某种技术进行生动形象的介绍。如果引入一个用户熟悉的东西，相信会使得我们更加容易同用户沟通。打个比方，我们要向外国朋友介绍中华民族的图腾——龙，它长得什么样？我们不妨说，它长着蛇一样的身子，身上有鳞甲，就像鱼鳞……三言两语，就说清楚了。图形界面设计中也是这样。

图 13-10 演示了最优化技术中的旅行商人问题。商人需要到许多城市做生意，现在给出各个城市的位置，需要选择一条路线，使得商人到达各个城市并且总路程最短。实际上，城市是由一个平面上随机分布的一些点来决定的。我们以美国地图的轮廓为背景，很容易地让用户熟悉该问题的应用背景。

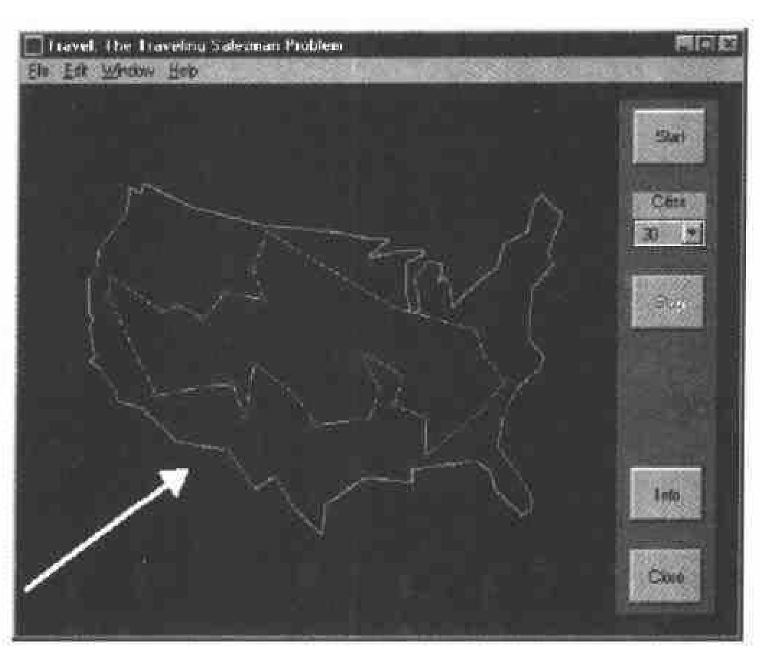


图 13-10　最短路径问题

下面，再介绍一个例子。该例子也是从生活中我们所熟悉的事物出发的。图 13-11 是用抛掷硬币来解释随机数据的产生。左上图的那枚硬币与上例中的美国地图轮廓有异曲同工之妙。

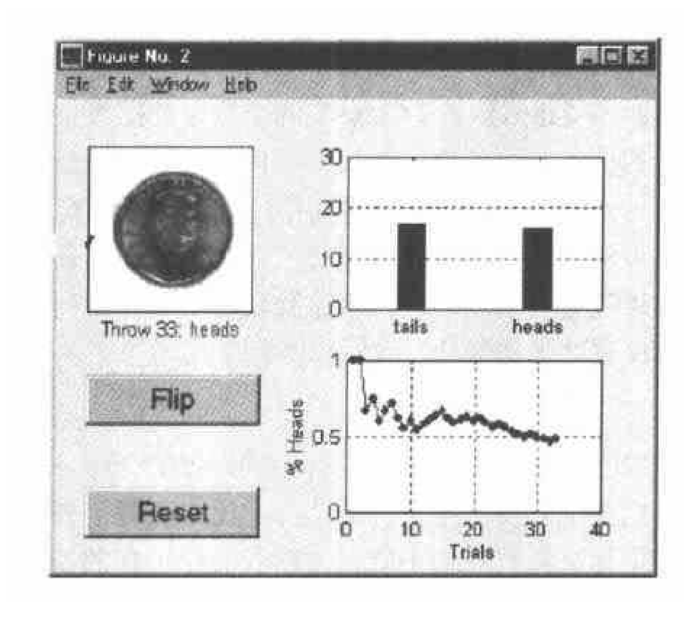


图 13-11　随机事件分布

13.3.2　动态界面的设计

目前我们的讨论主要偏向于静态界面的设计。当然，MATLAB 的 GUI 并不仅限于此。通过它，设计者也可以开发出生动活泼的动态界面。在设计动态界面的时候，除了应遵循以上所强调的几点原则外，还要注重下述几点：即时性，连续性，可逆性。

图 13-12 选自 MATLAB 的信号处理工具箱。为了观察傅里叶变换中，信号的频域和时域表示之间的对应关系，采用了动态界面的技术。通过拖拉滚动条或用鼠标直接拖拉波形，用户可直接完成波形频率和幅值的修改。

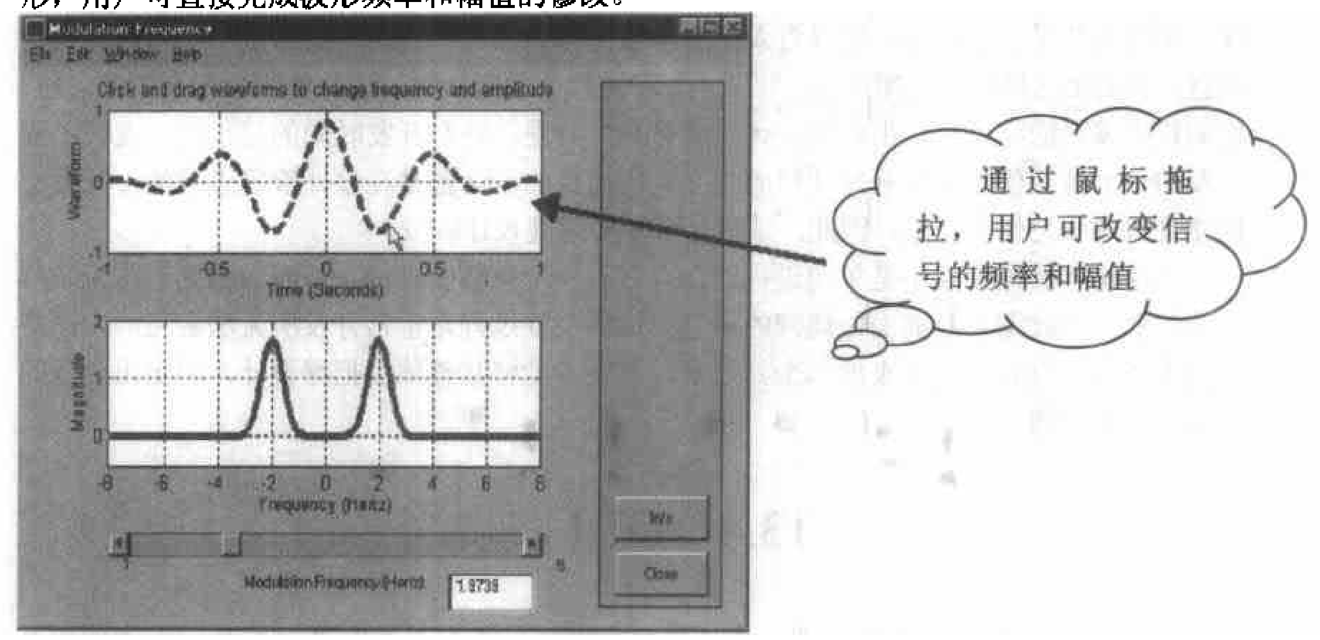


图 13-12 傅里叶变换演示

在设计动态界面的时候，设计者需要考虑计算量的大小。如果数据计算量太大，就会引起界面对用户输入响应的延时，达不到动态界面即时性与连续性的要求。这时，设计者必须考虑其他交互途径，比如用按钮来激发下一步操作。

13.3.3 开发流程

为了使读者对 MATLAB 的 GUI 开发过程有一个清晰的思路，这里给出了一个开发流程图，如图 13-13 所示。

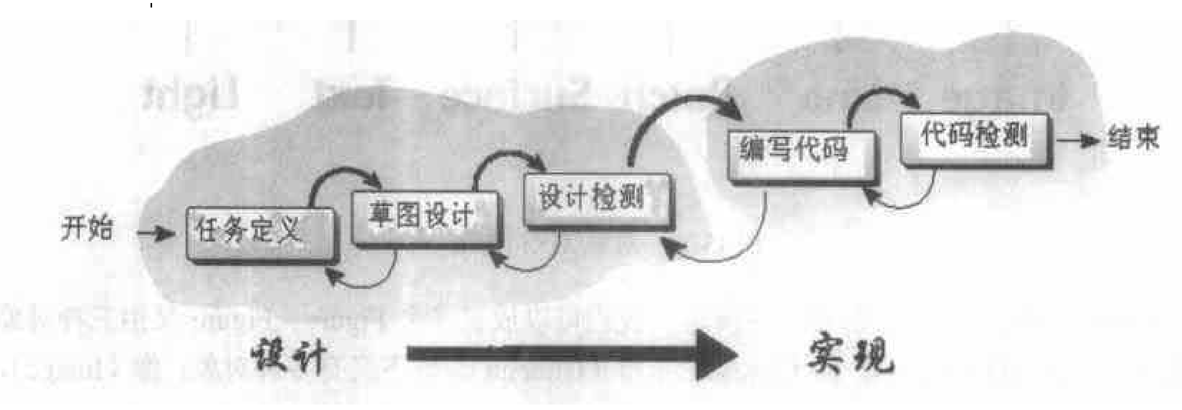


图 13-13 GUI 开发流程图

从图 13-13 可以看出，整个开发过程分为两个部分：设计阶段和实现阶段。一个好的设计是成功的一半，再一次提醒读者对设计予以足够的重视。很多人喜欢一下进入实现阶

段。在他们看来，设计阶段解决不了什么问题——首先，设计限制了后面的实现。在编写代码时，会有更多更好的想法蹦出来，这些想法会因设计的限制而被枪毙。其次，设计时考虑问题不会面面俱到，后面实现时会遇到一些无法克服的困难，不得不反过头来修改设计，因而浪费了大量时间。所以有人更愿意直接编写代码。他们的信念是——船到桥头自然直。表面上这种想法无懈可击。实际上，在进行工程实践时，这种做法会带来很多问题，比如代码编写的随意性，由此引发代码的可靠性问题，还有开发时间的延长等。现代工程开发的一大特征便是重视开发文档的建立。创造性的劳动都是在设计阶段进行的，开发文档便是对这些思想的记录。因此，请设计者不要忽视设计阶段。

请注意，开发流程不是单向进行的，有时有一个反复的过程。后面会遇到一些无法解决的问题，因此需要对前面做局部的调整。这种反复过程是正常开发所无法避免的。严格按照上述流程进行开发带来的一个好处是，开发者能够从整体上把握设计，制定出合适的策略来解决问题。

13.4 GUI 实现

在开始介绍编程实现 GUI 前，先让我们回顾一下第 1 章的内容。在 Figure 上的所有东西都是句柄图形的一个实例（见图 13-14）。它们都有一个独一无二的句柄。我们正是通过这个句柄把它们区分开来，从而改变每个对象的属性。

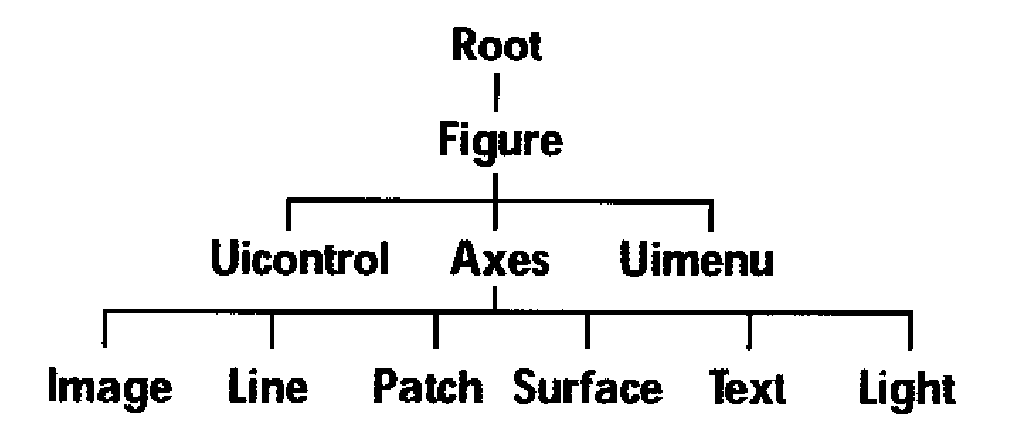


图 13-14 对象层次结构图

Root 可以看做是整个屏幕，在屏幕上我们可以放置多个 Figure。Figure 又由三种对象所组成：控件（Uicontrol），轴（Axes），菜单（Uimenu）。轴下又有多种对象：像（Image），线（Line），块（Patch），面（Surface），文本（Text），光（Light）。

看到上述介绍，读者也许会产生一大堆问题：如何获得这些对象的句柄？各种对象存在着多种属性，如何对各种属性进行设置？本节首先仔细介绍 GUIDE 的各种工具的使用，然后借助这些工具解决一些工程实例。希望这些工程实例能够给读者朋友的实际工作以帮助。

本节的体系结构是这样的，第一个实例侧重于介绍 GUIDE 工具的使用，后面的部分从 MATLAB 的各种应用环境中，挑选出一些有代表性的工程实例。首先叙述实例的应用背景，接着介绍开发 GUI 用到的主要技术。为了帮助读者进一步了解代码结构，书中还对部分核心代码做了剖析，并加上了翔实的注释。最后，列举了一些重要的函数命令以备读者检索。

13.4.1　实例一

该实例介绍 GUIDE 开发实例。

1．应用背景

用户通过按钮、列表框、可编辑文本框完成交互，实现对一个三维图形的控制，如图 13-15 所示。

2．目的

介绍 GUIDE 创建图形界面的开发步骤。如何利用控制面板（Control Panel）向图形上添加控件，如何利用属性编辑器（Property Editor）修改图形上各种对象的图形属性，如何利用回调函数编辑器（Callback Editor）编写回调函数来响应用户。这些都是最基本的操作。

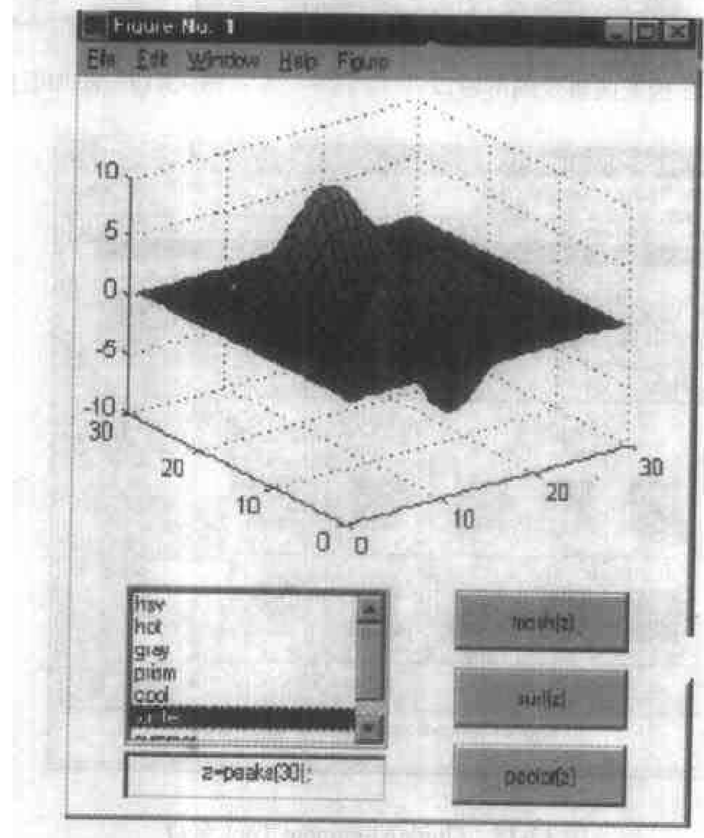


图 13-15　三维图形的控制图

3．创建步骤

（1）在 MATLAB 命令行中，运行指令 GUIDE。出现如图 13-16 所示的窗口。左上

方是控制面板，右下方是一个新的图形（Figure），我们所要做的工作都在这个图形上进行。

（2）选中控制面板上新对象栏中的第二排第一个按钮，在 Figure 上的合适区域画出坐标系。同样的步骤，可以将按钮、列表框、可编辑文本框加到 Figure 上。于是得到如图 13-17 所示的界面。

（3）下面需要打开 GUIDE 中的 ALIGNMENT 工具，调整界面上的各个控件的间距，使得界面显得整齐划一。具体方法是，单击控制面板菜单中的【Tools】命令，从弹出的菜单中单击【Alignment tool】命令，会出现如图 13-18 所示的界面。

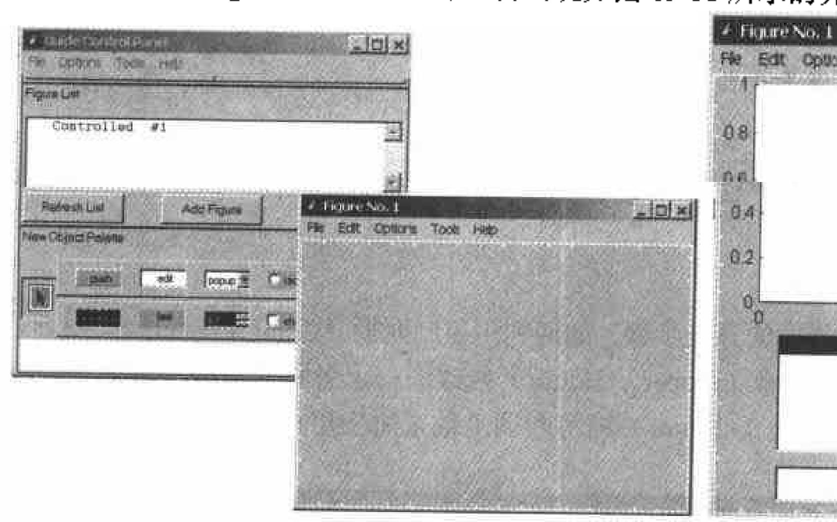


图 13-16　GUIDE 控制面板及图形窗口

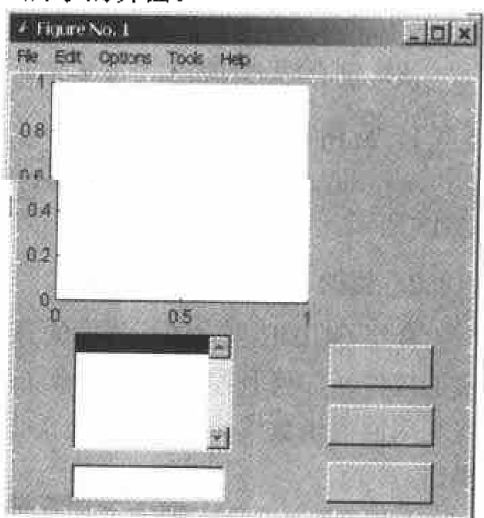


图 13-17　添加组件到图形上

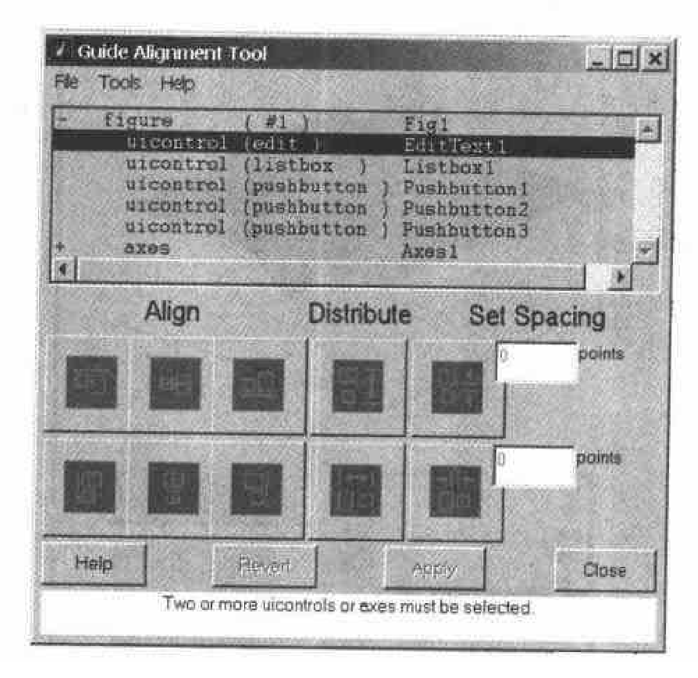


图 13-18　GuideAlignment Tool 窗口

注意，目前“Guide Alignment Tool”窗口中下面的所有工具都是灰色，处于无效状态。只有从上面列表中选取多个对象后，这些工具才会转为白色，即为有效状态。选取多个对象的方法是，在选择对象时，按住【Ctrl】键不放。接着，我们便可以使用“Align”工具



组中的工具使得多个对象向下对齐、向上对齐、向左对齐、向右对齐，或者使用“Distribute”工具组中的工具调整多个对象的上下间距、左右间距。

我们已完成了界面的布置，下面需要向列表框中添加选项。

（4）在 MATLAB 命令行中输入：

```
cmaps = {'hsv';'hot';'gray';'prism';'cool';'winter';'summer'};
```

于是，MATLAB 的工作空间中会生成一个字符串数组 cmaps。使用属性编辑器修改该列表框的字符串属性为 cmaps。注意不要加引号！如图 13-19 所示。

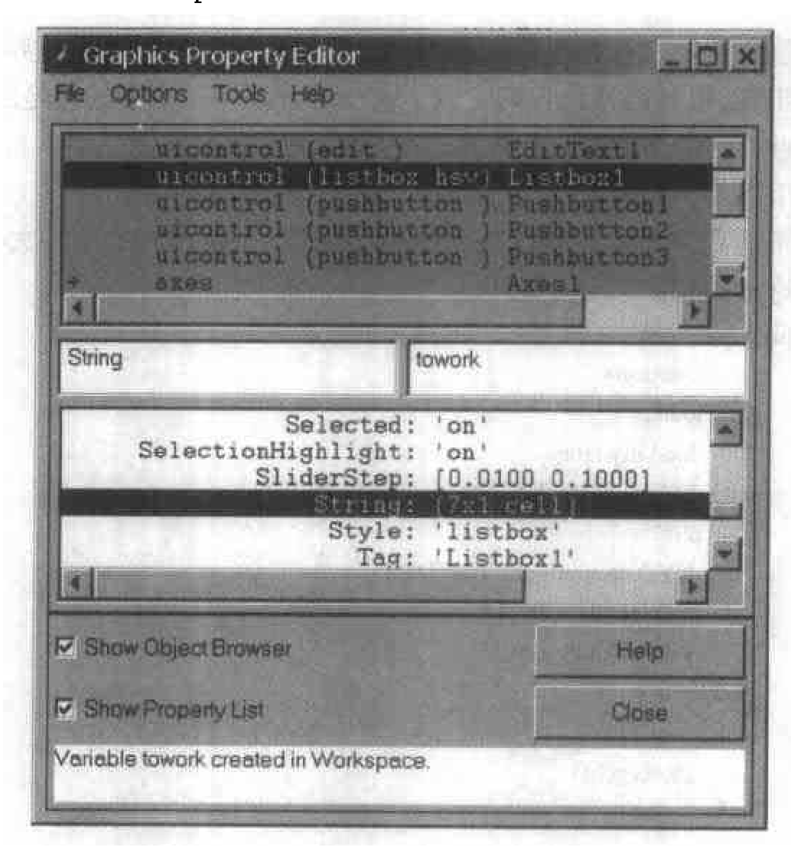


图 13-19　Graphics Property Editor 窗口

（5）打开回调函数编辑器，为列表框和按钮添加回调函数。先选中对象，接着选择函数类型“callback”，然后在编辑框中输入下列指令：

```
Value = get(gcbo,'Value');
                %返回列表框的值
String = get(gcbo,'String');
                %返回列表框的字符串
colormap(String{Value})
                %根据列表框的值和字符串，改变 Figure 的色表
```

同样，也可以为按钮添加回调函数。代码如下：

```
CommandString = get(gcbo,'String');
                %得到按钮所对应的画图指令
EditHandle = findobj(gcbf,'Tag','EditText1');
                %得到 GUIDE 分配给可编辑框的句柄
```

```
ZString = get(EditHandle,'String');
                    %从编辑框中获得指令
eval(ZString);
                    %执行编辑框中的指令
eval(CommandString);
                    %执行按钮所对应的指令
```

现在这个 GUI 就可以工作了。在对它测试之前，记住要将其保存。否则将功亏一篑！

4．回调函数的编写

正如我们把整个开发过程分为设计和实现，我们也可以把实现分为界面的布置和回调函数的编写。GUIDE 很好地帮助我们完成了前一部分工作，剩下的便是回调函数的编写。如果注意以下几点，回调函数的运行效率会更高。

- 尽量使用短小的函数来实现回调，而不是一长串字符串。
- 采用如下的程序结构，会增加程序的可读性。MATLAB 中的大部分示例程序都使用了这种结构。

```
function mygui(action)
        switch(action)
        case 'load',
            load mydata;
        set(gcbf,'UserData',XYData)
        case 'plot'
            XYData=get(gcbf,'UserData')
            x=XYData(:,1);
            y=XYData(:,2);
        plot(x,y)
        case 'close'
            close(gcbf)
end
```

说明

在回调函数内部，需要改变自身状态或其父对象的状态时，可以使用 gcbo 和 gcbf.注意 findobj 指令，该指令可以帮助我们减少大量的冗余变量。比较下面两段代码。

代码一：

```
lineHandle = plot(rand(10,1))
% 记住句柄，以备后面使用
……大量代码……
set(lineHandle,'LineWidth',4,'Color','red')
```

代码二：

```
plot(rand(10,1),'Tag','My Line')
% 不用记住代码，这是给它赋一个 Tag
    ……中间大量代码……
lineHandle = findobj('Tag','My Line');
    %根据 Tag 找回所要操作对象的句柄
    set(lineHandle,'LineWidth',4,'Color','red')
```



13.4.2 实例二

这是一个边缘检测实例

1．应用背景

该示例来源于 MATLAB 中的 demo（见图 13-20）。它调用 EDGE 程序，应用不同的边缘检测方法对多个图像进行处理。用户可以通过弹出式按钮选择一幅图像及边缘检测方法，也可以设置边缘检测算法中的参数。参数集随着检测方法的不同而变动。当完成设置后，用户可以通过【Apply】按钮激活图像的处理，按照设定的方法和参数对图像进行处理。

默认状态下，EDGE 程序会自动选择适当的门限阀值，当然为了满足用户对图像处理的要求，也可以手工设定门限阀值。

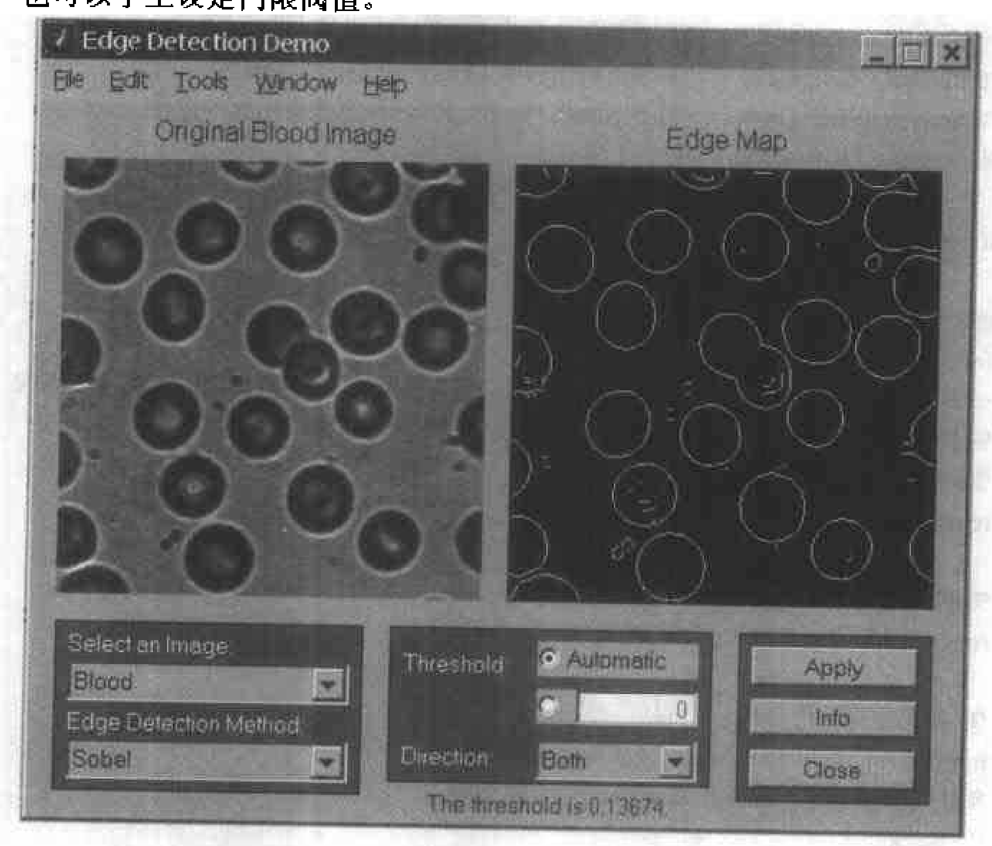


图 13-20 demo 程序

2．目的

使读者进一步熟悉各种控件的使用，并掌握复杂回调函数的编写。另外，该示例也介绍了如何根据用户的选择来动态更新界面的技术。

请读者注意，一个优美的界面是算法的漂亮外衣。如何给算法穿上这件外衣是我们设计者应该考虑的问题。毫无疑问，第一重要的是要量体裁衣。因此，界面是建立在算法基础之上的，算法对输入参数的要求应该转化为界面的语言清晰地向用户表达。

3．源代码

例程 13-2 是 demo 源代码。

例程 13-2

```
function edgedemo(action, varargin)
% 子函数:
% InitializeEDGEDEMO -  初始化图像、控件、轴
% ComputeEdgeMap -      调用 edge.m 计算原始图像的边缘图
% SelectMethod -        选取边缘检测算法，并使相应的控件有效或无效
% LoadNewImage -        读入选取的图像
% UpdateThreshCtrl -    从编辑框中获取门限值，并使【Apply】按钮有效
% UpdateDirectionality - 根据弹出菜单中的内容显示指示字符串
% Radio -               设置 Radio Buttons 中的值，并使得门限编辑框有效或无效
% UpdateLOGSize -       从编辑框中获取 LOG 过滤器的大小
% UpdateLOGSigma -      从编辑框中获取 LOG 过滤器的 Sigma 值
% ActivateSPRControls - 打开 Sobel, Prewitt, Roberts 方法中用到的控件
% ActivateLOGControls - 打开 LOG 方法中用到的控件

if nargin<1,
    action='InitializeEDGEDEMO';
end;
feval(action,varargin{:});
%注意，这里没有用前面推荐的程序结构，而是巧用 feval 函数来调用回调函数相应
%用户的界面操作，这也是一种比较常用的程序结构，往往用在回调函数比较复杂
%的情况下。
return;

%%%   Sub-function - InitializeEDGEDEMO
function InitializeEDGEDEMO()

% 如果程序已经运行，将它放到前台
h = findobj(allchild(0), 'tag', 'Edge Detection Demo');
% 0 代表根对象，在整个对象树中处于最顶层
if ~isempty(h)
    figure(h(1))
    return
end

screenD = get(0, 'ScreenDepth');
if screenD>8
    grayres=256;
else
    grayres=128;
end

%下面的代码主要完成对界面对象的描述，包括大小、位置及其属性
```



```
%由于篇幅所限，这里只给出 Figure 的设置
EdgeDemoFig = figure( ..
    'Name','Edge Detection Demo', ...
    'NumberTitle','off', 'HandleVisibility', 'on', ...
    'tag', 'Edge Detection Demo', ..
    'Visible','off', 'Resize', 'off',...
    'BusyAction','Queue','Interruptible','off', .
    'Color', [ 8 .8 .8], ...
    'IntegerHandle', 'off', .
    'DoubleBuffer', 'on', ...
    'Colormap', gray(grayres));

figpos = get(EdgeDemoFig, 'position');
% 调整 Figure 窗口的大小
figpos(3:4) = [560 420];
horizDecorations = 10;   % resize controls, etc
vertDecorations = 45;    % title bar, etc.
screenSize = get(0,'ScreenSize');
if (screenSize(3) <= 1)
    % No display connected (apparently)
    screenSize(3:4) = [100000 100000]; % 不要使用 Inf
end
if (((figpos(3) + horizDecorations) > screenSize(3)) | ..
            ((figpos(4) + vertDecorations) > screenSize(4)))
%对于这个 demo 来说，屏幕尺寸太小！
    delete(EdgeDemoFig);
    error(['Screen resolution is too low ', ...
            (or text fonts are too big) to run this demo']);
end
dx = screenSize(3) - figpos(1) - figpos(3) - horizDecorations;
dy = screenSize(4) - figpos(2) - figpos(4) - vertDecorations
if (dx < 0)
    figpos(1) = max(5,figpos(1) + dx);
end
if (dy < 0)
    figpos(2) = max(5,figpos(2) + dy);
end
set(EdgeDemoFig, 'position', figpos);

rows = figpos(4); cols = figpos(3);
hs = (cols-512) / 3;          % Horizantal Spacing
bot = rows-2*hs-256;          % Bottom of the images

%===================================
% 接下来是控件和菜单的设置，省略
    ......
```

```
%==================================

set(EdgeDemoFig, 'Userdata', hdl, 'Visible', 'on');
%初始化中创建的所有对象的句柄都放入 hdl 中，然后又将 hdl 放入图形 Figure
%的“Userdata”，由此实现了句柄的传递。这是一种很好的共享数据的方法，请读者
%注意
drawnow
LoadNewImage(EdgeDemoFig);
drawnow
set(EdgeDemoFig, 'HandleVisibility', 'Callback');
set([hdl.Apply hdl.Help hdl.Close] , 'Enable', 'on');
return

%%%
%%%   Sub-Function - ComputeEdgeMap
%%%

function ComputeEdgeMap(DemoFig)

if nargin<1
    callb = 1;
    DemoFig = gcbf;
else
    callb = 0;
end

set(DemoFig,'Pointer','watch');
setstatus(DemoFig, 'Computing the edge map...');
hdl=get(DemoFig,'Userdata');
img = getimage(hdl.Image);

autothresh = get(hdl.RadioAutomatic, 'Value');

switch hdl.Method
case {'Sobel','Roberts','Prewitt'}
    if autothresh
        [edgemap,thresh]=edge(img,hdl.Method, dl.Directionality);
        setstatus(['The threshold is ' num2str(thresh) '.']);
    else
        edgemap = edge(img, ...
            hdl.Method, hdl.Threshold, hdl.Directionality);
        setstatus(DemoFig, '');
    end
case 'Laplacian of Gaussian'
    if autothresh
        [edgemap,thresh] = edge(img, 'log', [], hdl.LogSigma);
```



```
        setstatus(DemoFig, ['The threshold is ' num2str(thresh) '.']);
    else
        edgemap = edge(img, 'log', hdl.Threshold, hdl.LogSigma);
        setstatus(DemoFig, '');
    end
case 'Canny'
    if autothresh
        [edgemap,thresh] = edge(img, 'canny', [], hdl.LogSigma);
        setstatus(DemoFig, ['High threshold is ' num2str(thresh(2)) '.']);
    else
        [edgemap,thresh] = edge(img, 'canny', hdl.Threshold, hdl.LogSigma);
        setstatus(DemoFig, '');
    end
otherwise
    error('EDGEDEMO: Invalid edge detection method.');
end

set(hdl.Edge, 'CData', edgemap);
set(hdl.Apply, 'Enable', 'off');
set(DemoFig,'Pointer','arrow');
drawnow


%%%
%%%    Sub-Function - SelectMethod
%%%

function SelectMethod

DemoFig = gcbf;

hdl = get(DemoFig, 'userdata');
v = get(hdl.MethodPop,{'value','String'});
hdl.Method = deblank(v{2}(v{1},:));
switch hdl.Method
case {'Sobel','Prewitt'}
    ActivateSPRControls(DemoFig);
    set(hdl.sprDirPop, 'Enable', 'on');
case 'Laplacian of Gaussian'
    ActivateLOGControls(DemoFig);
    set(hdl.logSigmaCtrl, 'String', '2');
    hdl.LogSigma = 2;
case 'Canny
    ActivateLOGControls(DemoFig);
    set(hdl.logSigmaCtrl, 'String', '1');
    hdl.LogSigma = 1;
```

```
case 'Roberts'
    ActivateSPRControls(DemoFig);
    set(hdl.sprDirPop, 'Enable', 'off', 'value', 1);
otherwise
    error('EDGEDEMO: invalid method specifier.');
end

set(hdl.Apply, 'Enable', 'on');
set(DemoFig, 'userdata', hdl);
% BlankOutEdgeMap(DemoFig);
setstatus(DemoFig, ['Press "Apply" to compute edges.']);


%%%
%%%   Sub-Function - LoadNewImage
%%%

function LoadNewImage(DemoFig)

if nargin<1
    callb = 1;      % We're in a callback
    DemoFig = gcbf;
else
    callb = 0;      % We're in the initialization
end

set(DemoFig,'Pointer','watch');
%由于图像加载时间可能比较长，在加载前把鼠标的形状改成沙漏形
hdl=get(DemoFig,'Userdata');
v = get(hdl.ImgPop,{'value','String'});
name = deblank(v{2}(v{1},:));
setstatus(DemoFig, ['Loading the ' name ' image...']);
drawnow

switch name
case 'Aluminum',
    alumgrns2 = [];                    % Parser hint
    load imdemos alumgrns2
%注意这里图像是放在工作空间 imdemos 中，所以使用了 load。否则，需要使用 imread
%函数。具体使用请参照前面的介绍
    img = alumgrns2;
case 'Blood',
    blood2 = [];
    load imdemos blood2
    img = blood2;
case 'Rice',
```

```
        rice3 = [];
        load imdemos rice3
        img = rice3;
    case 'Saturn',
        saturn2 = [];
        load imdemos saturn2
        img = saturn2;
    case 'Eight Bit',
        eight = [];
        load imdemos eight
        img = eight;
    case 'Circuit',
        circuit4 = [];
        load imdemos circuit4
        img = circuit4;
    case 'Vertigo',
        vertigo2 = [];
        load imdemos vertigo2
        img = vertigo2;
    case 'Bone Marrow',
        bonemarr2 = [];
        load imdemos bonemarr2
        img = bonemarr2;
    otherwise
        error('EDGEDEMO: Unknown Image Option!');
    end

    set(hdl.Image, 'Cdata', img);
    set(get(hdl.ImageAxes,'title'),'string',['Original ' name ' Image']);
    set(DemoFig,'Pointer','arrow');
    if callb
        set(hdl.Apply, 'Enable', 'on');
    end
    drawnow
    if ~strcmp(hdl.Method, 'Laplacian of Gaussian')
        if get(hdl.RadioAutomatic, 'Value')==0
            Radio('auto',DemoFig);
        end
    end
    ComputeEdgeMap(DemoFig);
    return;


    %%%
    %%%  Sub-function - UpdateThreshCtrl
    %%%
```

```
function UpdateSprThresh()
DemoFig = gcbf;
hdl = get(DemoFig, 'UserData');
v = hdl.Threshold;
s = get(hdl.ThreshCtrl,'String');
vv = real(evalin('base',['[' s ']'],num2str(v)));
%验证输入
if isempty(vv) | ~isreal(vv) | vv(1)<0
    vv = v;
    set(gcbo,'String',num2str(vv));
    return
end
vv = round(vv(1)*1000000)/1000000;
set(gcbo,'String',num2str(vv));
hdl.Threshold = vv;
set(hdl.Apply, 'Enable', 'on');
setstatus(DemoFig, 'Press "Apply" to compute edges.');
set(DemoFig, 'UserData', hdl);
return

%%%
%%%   Sub-function - UpdateDirectionality
%%%

function UpdateDirectionality()

DemoFig = gcbf;
hdl = get(DemoFig, 'UserData');
v = get(hdl.sprDirPop,{'value','String'});
dir = deblank(v{2}(v{1},:));
set(hdl.sprDirPop, 'userdata', dir);
hdl.Directionality = dir;
set(hdl.Apply, 'Enable', 'on');
setstatus(DemoFig, 'Press "Apply" to compute edges.');
set(DemoFig, 'UserData', hdl);
return


%%%
%%% Sub-function - Radio
%%%

unction Radio(control, DemoFig)

if nargin<2
```

```
    DemoFig = gcbf;
end
hdl = get(DemoFig, 'UserData');
if strcmp(control, 'auto')
    set(hdl.RadioAutomatic, 'Value', 1);
    set(hdl.RadioManual, 'Value', 0);
    set(hdl.ThreshCtrl, 'Enable', 'off');
    set(hdl.Apply, 'Enable', 'on');
    setstatus(DemoFig, 'Press "Apply" to compute edges.');
elseif strcmp(control, 'manual')
    set(hdl.RadioAutomatic, 'Value', 0);
    set(hdl.RadioManual, 'Value', 1);
    set(hdl.ThreshCtrl, 'Enable', 'on');
    set(hdl.Apply, 'Enable', 'on');
    setstatus(DemoFig, 'Press "Apply" to compute edges.');
end
return


%%%
%%% Sub-function - UpdateLOGSigma
%%%当用户输入 SIGMA 值后，一方面需要验证取值的有效性，
%%%另一方面需要更新界面

function UpdateLOGSigma()
DemoFig = gcbf;
hdl = get(DemoFig, 'UserData');
v = hdl.LogSigma;
s = get(hdl.logSigmaCtrl,'String');
vv = real(evalin('base',s,num2str(v)));
if isempty(vv) | ~isreal(vv) | vv(1)<0
% 验证输入的 SIGMA 值是否有效
    vv = v;
    set(hdl.logSigmaCtrl,'String',num2str(vv));
    return
end
vv = round(vv(1)*100)/100;
set(hdl.logSigmaCtrl,'String',num2str(vv));
hdl.LogSigma = vv;
set(hdl.Apply, 'Enable', 'on');
setstatus(DemoFig, 'Press "Apply" to compute edges.');
set(DemoFig, 'UserData', hdl);
return

% 下面两个子程序都是通过改变控件的可见属性来完成界面的切换
%%%
```

```
%%% Sub-function - ActivateSPRControls
%%%该子程序激活 SPR 三种算法使用的公共控件

function ActivateSPRControls(DemoFig)
hdl = get(DemoFig, 'UserData');
set([hdl.sprDirPop hdl.sprDirLbl], 'Visible', 'on');
set([hdl.logSigmaCtrl hdl.logSigmaLbl], 'Visible', 'off');

%%%
%%%Sub-function – ActivateLOGControls
%%%该子程序激活 LOG 算法中使用的控件

function ActivateLOGControls(DemoFig)
hdl = get(DemoFig, 'UserData');
set([hdl.logSigmaCtrl hdl.logSigmaLbl],'Visible', 'on');
set([hdl.sprDirPop hdl.sprDirLbl], 'Visible', 'off');
```



在该实例中，用户需注意以下两点：

第一，使用 hdl 结构体来保存图形界面中各种对象的句柄 这些句柄在创建图形对象时产生，并在以后的回调函数中多次用到.

第二，各个对象中的“userdata”属性域的妙用。我们可以通过它在子程序之间传递数据，这样做避免了全局变量的使用。

附录 A　图像处理工具箱函数详解

图像显示

1．colorbar

功能：

显示颜色条

语法：

```
colorbar('vert')
colorbar('horiz')
colorbar(h)
h=colorbar(…)
```

举例：

```
surf(peaks(30))
colormap cool
colorbar
```

参考： colormap

2．Getimage

功能：

从坐标轴获得图像数据

语法：

```
A=getimage(h)
[x,y,A]=getimage(h)
[…,A,flag]=getimage(h)
[…]=getimage
```

举例：

```
imshow rice.tif
I = getimage;
```

参考： 无

3．imshow

功能：

显示图像

语法：

```
imshow (I,n)
imshow (I,[low high])
imshow (RW)
```

```
imshow (X,map)
imshow(RGB)
imshow(…,display_option)
imshow(x,y,A, …)
imshow filename
h= imshow(…)
```

举例：

无

参考： getimage, imread, iptgetpref, iptsetpref, subimage, truesize, warp

4．montage

功能：

在矩形框中同时显示多幅图像

语法：

```
montage(I)
montage(BW)
montage(X,map)
montage(RGB)
h=montage(…)
```

举例：

```
load mri
montage(D,map)
```

参考： immovie

5．immovie

功能：

创建多帧索引图的电影动画

语法：

```
mov=immovie(X,map)
```

举例：

```
load mri
mov = immovie(D,map);
movie(mov,3)
```

参考： avifile, getframe, montage, movie, movie2avi

6．subimage

功能：

在一幅图中显示多幅图像

语法：

```
subimage(X,map)
subimage(I)
subimage(BW)
subimage(RGB)
subimage(x,y, …)
```

```
h=subimage(…)
```

举例：

```
load trees
[X2,map2] = imread('forest.tif');
subplot(1,2,1), subimage(X,map)
subplot(1,2,2), subimage(X2,map2)
```

参考：imshow ，subplot

7．truesize

功能：

调整图像显示尺寸

语法：

```
truesize(fig)
truesize(fig,[mrows mcols])
```

举例：

无

参考：imshow, iptsetpref, iptgetpref

8．warp

功能：

将图像显示到纹理映射表面

语法：

```
warp(X,map)
warp(I,n)
warp(BW)
warp(RGB)
warp(z,…)
warp(x,y,z,…)
h=warp(…)
```

举例：

```
[x,y,z] = cylinder;
I = imread('testpat1.tif');
warp(x,y,z,I);
```

参考：imshow，image, imagesc, surf

9．zoom

功能：

缩放图像

语法：

```
zoom on
zoom off
zoom out
zoom reset
zoom
```

```
zoom xon
zoom yon
zoom(factor)
zoom(fig,option)
```

举例：

无

参考： 无

图像文件 I/O

1．imfinfo

功能：

返回图形文件信息

语法：

```
info= imfinfo(filename,fmt)
info= imfinfo(filename)
```

举例：

```
info = imfinfo('canoe.tif')
```

参考： imformats, imread, imwrite

2．imread

功能：

从图形文件中读取图像

语法：

```
A= imread(filename,fmt)
[X,map]= imread(filename,fmt)
[…]= imread(filename)
[…]= imread(…,idx) (TIFF only)
[…]= imread(…,ref) (HDF only)
[…]= imread(…,'BackgroundColor',BG) (PNG only)
[A,map,alpha]= imread(…) (PNG only)
```

举例：

```
[a,b,c] = imread('your_icon.ico');
% Augment colormap for background color (white).
b2 = [b; 1 1 1];
% Create new image for display.
d = ones(size(a)) * (length(b2) - 1);
% Use the AND mask to mix the background and
% foreground data on the new image
d(c == 0) = a(c == 0);
% Display new image
image(uint8(d)), colormap(b2)
```

参考： double, fread, imfinfo, imformats, imwrite, uint8, uint16

3．imwrite

功能：

将图像写入到图形文件中

语法：

```
imwrite(A,filename,fmt)
imwrite(X,map,filename,fmt)
imwrite(…,filename)
imwrite(…,param1,var1,param2,var2,…)
```

举例：

```
imwrite(X,map,'your_hdf_file.hdf','Compression','none','WriteMode','append')
```

参考： fwrite, imfinfo, imformats, imread

几何操作

1．imcrop

功能：

剪切图像

语法：

```
I2= imcrop(I)
X2= imcrop(X,map)
RGB2= imcrop(RGB)
I2= imcrop(I,rect)
X2= imcrop(X,map,rect)
RGB2= imcrop(RGB,rect)
[…]= imcrop(x,y,…)
[A,rect]= imcrop(…)
[x,y,A,rect]=imcrop(…)
```

举例：

```
I = imread('ic.tif');
I2 = imcrop(I,[60 40 100 90]);
imshow(I)
figure, imshow(I2)
```

参考： zoom

2．imresize

功能：

改变图像大小

语法：

```
B= imresize(A,m,method)
B= imresize(A,[mrows ncols],method)
B= imresize(…,method,n)
B= imresize(…,method,h)
```

举例：

无

参考： imrotate, imtransform, tformarray; interp2

3．imrotate

功能：

旋转图像

语法：

```
B= imrotate(A,angle,method)
B= imrotate(A,angle,method,'crop')
```

举例：

```
I = imread('ic.tif');
J = imrotate(I,-4,'bilinear','crop');
imshow(I)
figure, imshow(J)
```

参考： imcrop, imresize, imtransform, tformarray

像素和统计处理

1．corr2

功能：

计算两个矩阵的二维相关系数

语法：

```
r=corr2(A,B)
```

举例：

无

参考： std2 corrcoef

2．imcontour

功能：

创建图像数据的轮廓图

语法：

```
imcontour(I,n)
imcontour(I,v)
imcontour(x,y,…)
imcontour(…,LineSpec)
[C,h]= imcontour(…)
```

举例：

```
I = imread('ic.tif');
imcontour(I,3)
```

参考： clabel, contour, LineSpec

3．imfeature

功能：

计算图像区域的特征尺寸

语法：

```
stats= imfeature(L,measurements)
stats= imfeature(L,measurements,n)
```

举例：

```
BW = imread('text.tif');
L = bwlabel(BW);
stats = imfeature(L,'all');
stats(23)
```

参考： bwlabel，ismember

4．imhist

功能：

显示图像数据的柱状图

语法：

```
imhist(I,n)
imhist(X,map)
[counts,x]=imhist(…)
```

举例：

```
I = imread('pout.tif');
imhist(I)
```

参考： histeq，hist

5．impixel

功能：

确定像素颜色值

语法：

```
P= impixel(I)
P= impixel(X,map)
P= impixel(RGB)
P= impixel(I,c,r)
P= impixel(X,map,c,r)
P= impixel(RGB,c,r)
[c,r,P]= impixel(…)
P= impixel(x,y,I,xi,yi)
P= impixel(x,y,X,map,xi,yi)
P= impixel(x,y,RGB,xi,yi)
[xi,yi,P]= impixel(x,y,…)
```

举例：

```
RGB = imread('flowers.tif');
c = [12 146 410];
r = [104 156 129];
```

```
pixels = impixel(RGB,c,r)
```

参考：improfile, pixval

6．improfile

功能：

沿线段计算剖面图的像素值

语法：

```
c= improfile
c= improfile(a)
c= improfile(I,xi,yi)
c= improfile(I,xi,yi,n)
[cx,cy,c]= improfile(…)
[cx,cy,c,xi,yi]= improfile(…)
[…]= improfile(x,y,I,xi,yi)
[…]= improfile(x,y,I,xi,yi,n)
[…]= improfile(…,method)
```

举例：

```
I = imread('alumgrns.tif');
x = [35 338 346 103];
y = [253 250 17 148];
improfile(I,x,y), grid on
```

参考：impixel, pixval, interp2

7．mean2

功能：

计算矩阵元素的平均值

语法：

```
b= mean2(A)
```

举例：

无

参考：std2 ,mean, std

8．pixval

功能：

显示图像像素的信息

语法：

```
pixval on
pixval off
pixval
pixval(fig,option)
```

举例：

无

参考：impixel, improfile

9．std2

功能：

计算矩阵元素的标准偏移

语法：

```
b=std2(A).
```

举例：

无

参考： corr2, mean2，std, mean

图像分析

1．edge

功能：

识别强度图像中的边界

语法：

```
BW= edge(I,'sobel')
BW= edge(I,'sobel',thresh)
BW= edge(I,'sobel',thresh,direction)
[BW,thresh]= edge(I,'sobel',…)
BW= edge(I,'prewitt')
BW= edge(I,'prewitt',thresh)
BW= edge(I,'prewitt',thresh,direction)
[BW,thresh]= edge(I,'prewitt',…)
BW= edge(I,'roberts')
BW= edge(I,'roberts',thresh)
[BW,thresh]= edge(I,'roberts',…)
BW= edge(I,'log')
BW= edge(I,'log',thresh)
BW= edge(I,'log',thresh,direction)
[BW,threshold]= edge(I,'log',…)
BW= edge(I,'zerocross',thresh,h)
[BW,thresh]= edge(I,'zerocross',…)
BW= edge(I,'canny')
BW= edge(I,'canny',thresh)
BW= edge(I,'canny',thresh,sigma)
[BW,threshold]= edge(I,'canny',…)
```

举例：

```
I = imread('rice.tif');
BW1 = edge(I,'prewitt');
BW2 = edge(I,'canny');
imshow(BW1);
figure, imshow(BW2)
```

参考： fspecial

2．qtdecomp

功能：

进行四叉树分解

语法：

```
S= qtdecomp(I)
S= qtdecomp(I,threshold)
S= qtdecomp(I,threshold,mindim)
S= qtdecomp(I,threshold,[mindim maxdim])
S= qtdecomp(I,fun)
S= qtdecomp(I,fun,P1,P2,…)
```

举例：

```
I = [1    1    1    1    2    3    6    6
     1    1    2    1    4    5    6    8
     1    1    1    1    10   15   7    7
     1    1    1    1    20   25   7    7
     20   22   20   22   1    2    3    4
     20   22   22   20   5    6    7    8
     20   22   20   20   9    10   11   12
     22   22   20   20   13   14   15   16];
S = qtdecomp(I,5);
full(S)
```

参考： qtgetblk, qtsetblk

3．qtgetblk

功能：

获取四叉树分解中的块值

语法：

```
[vals, r,c,]= qtgetblk(I,S,dim)
[vals,idx]= qtgetblk(I,S,dim)
```

举例：

```
[vals,r,c] = qtgetblk(I,S,4)
```

参考： qtdecomp, qtsetblk

4．qtsetblk

功能：

设置四叉树分解中的块值

语法：

```
J= qtsetblk (I,S,dim,vals)
```

举例：

```
newvals = cat(3,zeros(4),ones(4));
J = qtsetblk(I,S,4,newvals)
```

参考： qtdecomp, qtgetblk



图像增强

1．histeq

功能：

用柱状图均等化增强对比

语法：

```
J= histeq(I,hgram)
J= histeq(I,n)
[J,T]= histeq(I,…)
```

举例：

```
I = imread('tire.tif');
J = histeq(I);
imshow(I)
figure, imshow(J)
```

参考： brighten, imadjust, imhist

2．imadjust

功能：

调整图像灰度值或颜色映像表

语法：

```
J= imadjust(I,[low high],[bottom top],gramma)
newmap= imadjust(map,[low high],[bottom top],gramma)
RGB2= imadjust(RGB,…)
```

举例：

```
I = imread('pout.tif');
J = imadjust(I,[0.3 0.7],[]);
imshow(I), figure, imshow(J)
```

参考： brighten, histeq, stretchlim

3．imnoise

功能：

增强图像的渲染效果

语法：

```
J= imnoise(I,type)
J= imnoise(I,type,parameters)
```

举例：

```
I = imread('eight.tif');
J = imnoise(I,'salt & pepper',0.02);
imshow(I)
figure, imshow(J)
```

参考： rand, randn

4．medfilt2

功能：

进行二维中值滤波

语法：

```
B= medfilt2(A,[m n])
B= medfilt2(A)
B= medfilt2(A,'indexed',…)
```

举例：

```
I = imread('eight.tif');.
J = imnoise(I,'salt & pepper',0.02);
K = medfilt2(J);
imshow(J)
figure, imshow(K)
```

参考：filter2, ordfilt2, wiener2

5．ordfilt2

功能：

进行二维统计顺序滤波

语法：

```
B= ordfilt2(A,order,domain)
B= ordfilt2(A,order,domain,S)
B= ordfilt2(…,padopt)
```

举例：

无

参考：filter2, ordfilt2, wiener2

6．wiener2

功能：

进行二维适应性去噪过滤处理

语法：

```
J= wiener2(I,[m n],noise)
[J,noise]= wiener2(I,[m n])
```

举例：

```
I = imread('saturn.tif');
J = imnoise(I,'gaussian',0,0.005);
K = wiener2(J,[5 5]);
imshow(J)
figure, imshow(K)
```

参考：filter2, medfilt2



线性滤波

1．conv2

功能：

进行二维卷积

语法：

```
C= conv2(A,B)
C= conv2(hcol,hrow,A)
C= conv2(…,shape)
```

举例：

```
A = rand(3);
B = rand(4);
C = conv2(A,B)                %C 是 6×6 的
```

参考： conv, convn, filter2

2．convmtx2

功能：

计算二维卷积矩阵

语法：

```
T= convmtx2(H,m,n)
T= convmtx2(H,[m n])
```

举例：

无

参考： conv2

3．convn

功能：

计算 n 维卷积

语法：

```
C= convn(A,B)
C= convn(A,B,shape)
```

举例：

无

参考： conv, conv2

4．filter2

功能：

进行二维线性过滤操作

语法：

```
B= filter2(h,A)
B= filter2(h,A,shape)
```

举例：

无

参考：conv2, filter

5．fspecial

功能：

创建预定义过滤器

语法：

```
h= fspecial(type)
h= fspecial(type,parameters)
```

举例：

```
I = imread('saturn.tif');
subplot(2,2,1);imshow(I);title('Original Image');
H = fspecial('motion',50,45);
MotionBlur = imfilter(I,H);
subplot(2,2,2);imshow(MotionBlur);title('Motion Blurred Image');
H = fspecial('disk',10);
blurred = imfilter(I,H);
subplot(2,2,3);imshow(blurred);title('Blurred Image');
H = fspecial('unsharp');
sharpened = imfilter(I,H);
subplot(2,2,4);imshow(sharpened);title('Sharpened Image');
```

参考：conv2, edge, filter2, fsamp2, fwind1, fwind2, imfilter，del2

线性二维滤波设计

1．freqspace

功能：

确定二维频率响应的频率空间

语法：

```
[f1,f2]= freqspace(n)
[f1,f2]= freqspace([m n])
[x1,y1]= freqspace(…,'meshgrid')
f= freqspace(N)
f= freqspace(N,'whole')
```

举例：

无

参考：meshgrid

2．freqz2

功能：

计算二维频率响应

语法：



```
[H,f1,f2]= freqz2(h,n1,n2)
[H,f1,f2]= freqz2(h,[n2 n1])
[H,f1,f2]= freqz2(h,f1,f2)
[H,f1,f2]= freqz2(h)
[…]= freqz2(h,…,[dx dy])
[…]= freqz2(h,…,dx)
freqz2(…)
```

举例：

```
Hd = zeros(16,16);
Hd(5:12,5:12) = 1;
Hd(7:10,7:10) = 0;
h = fwind1(Hd,bartlett(16));
colormap(jet(64))
freqz2(h,[32 32]); axis ([-1 1 -1 1 0 1])
```

参考： freqz

3．fsamp2

功能：

用频率采样法设计二维 FRI 滤波器

语法：

```
h= fsamp2(Hd)
h= fsamp2(f1,f2,Hd,[m n])
```

举例：

```
[f1,f2] = freqspace(21,'meshgrid');
Hd = ones(21);
r = sqrt(f1.^2 + f2.^2);
Hd((r<0.1)|(r>0.5)) = 0;
colormap(jet(64))
mesh(f1,f2,Hd)
h = fsamp2(Hd);
freqz2(h)
```

参考： conv2, filter2, freqspace, ftrans2, fwind1、fwind2

4．ftrans2

功能：

通过频率转换设计二维 FRI 滤波器

语法：

```
h= ftrans2(b,t)
h= ftrans2(b)
```

举例：

```
colormap(jet(64))
b = remez(10,[0 0.05 0.15 0.55 0.65 1],[0 0 1 1 0 0]);
[H,w] = freqz(b,1,128,'whole');
plot(w/pi-1,fftshift(abs(H)))
```

```
h = ftrans2(b);
freqz2(h)
```

参考：conv2, filter2, fsamp2, fwind1, fwind2

5. fwind1

功能：

用一维窗口方法设计二维 FRI 滤波器

语法：

```
h= fwind1(Hd,win)
h= fwind1(Hd,win1,win2)
h= fwind1(f1,f2,Hd,…)
```

举例：

```
[f1,f2] = freqspace(21,'meshgrid');
Hd = ones(21);
r = sqrt(f1.^2 + f2.^2);
Hd((r<0.1)|(r>0.5)) = 0;
colormap(jet(64))
mesh(f1,f2,Hd)
h = fwind1(Hd,hamming(21));
freqz2(h)
```

参考：conv2, filter2, fsamp2, freqspace, ftrans2, fwind2

6. fwind2

功能：

用二维窗口方法设计二维 FRI 滤波器

语法：

```
h= fwind2(Hd,win)
h= fwind2(f1,f2,Hd,win)
```

举例：

```
[f1,f2] = freqspace(21,'meshgrid');
Hd = ones(21);
r = sqrt(f1.^2 + f2.^2);
Hd((r<0.1)|(r>0.5)) = 0;
colormap(jet(64))
mesh(f1,f2,Hd)
win = fspecial('gaussian',21,2);
win = win ./ max(win(:));
mesh(win)
h = fwind2(Hd,win);
freqz2(h)
```

参考：conv2, filter2, fsamp2, freqspace, ftrans2, fwind1

图像变换

1．dct2

功能：

进行二维余弦变换

语法：

```
B= dct2(A)
B= dct2(A,m,n)
B= dct2(A,[m n])
```

举例：

```
RGB = imread('autumn.tif');
I = rgb2gray(RGB);
J = dct2(I);
imshow(log(abs(J)),[]), colormap(jet(64)), colorbar
```

参考： fft2, idct2, ifft2

2．dctmtx

功能：

计算离散余弦变换矩阵

语法：

```
D=dctmtx(n)
```

举例：

无

参考： dct2

3．fft2

功能：

进行二维快速傅里叶变换

语法：

```
B= fft2(A)
B= fft2(A,m,n)
```

举例：

无

参考： fft, fftn, fftshift, ifft2

4．fftn

功能：

进行 n 维快速傅里叶变换

语法：

```
B= fftn(A)
B= fftn(A,size)
```

举例：

无

参考：fft, fft2, fftn, ifftn

5．fftshift

功能：

把快速傅里叶变换的 DC 的组件移到频谱中心

语法：

```
B= fftshift(A)
```

举例：

```
Z = fftshift(Y)
```

参考：circshift, fft, fft2, fftn, ifftshift

6．idct2

功能：

二维离散反余弦变换

语法：

```
B= idct2(A)
B= idct2(A,m,n)
B= idct2(A,[m n])
```

举例：

无

参考：dct2, dctmtx, fft2, ifft2

7．ifft2

功能：

二维离散傅里叶变换

语法：

```
B= ifft2 (A)
B= ifft2 (A,m,n)
```

举例：

无

参考：fft2, fftshift, ifft, ifftn, ifftshift

8．ifftn

功能：

n 维离散傅里叶变换

语法：

```
B= ifftn (A)
B= ifftn (A,size)
```

举例：

无

参考：fftn, ifft, ifft2, ifftshift



9．iradon

功能：

进行反 radon 变换

语法：

```
I= iradon(P,theta)
I= iradon(P,theta,interp,filter,d,n)
[I,h]= iradon(…)
```

举例：

```
P = phantom(128);
R = radon(P,0:179);
I = iradon(R,0:179,'nearest','Hann');
imshow(P)
figure, imshow(I)
```

参考： radon, phantom

10．phantom

功能：

产生一个头部幻影图像

语法：

```
P= phantom(def,n)
P= phantom(E,n)
[P,E]= phantom(…)
```

举例：

```
P = phantom('Modified Shepp-Logan',200);
imshow(P)
```

参考： radon, iradon

11．radon

功能：

radon 变换

语法：

```
R= radon(I,theta)
R= radon(I,theta,n)
[R,xp]= radon(…)
```

举例：

```
iptsetpref('ImshowAxesVisible','on')
I = zeros(100,100);
I(25:75,25:75) = 1;
theta = 0:180;
[R,xp] = radon(I,theta);
imshow(theta,xp,R,[],'notruesize'), colormap(hot), colorbar
```

参考： iradon, phantom

边沿和块处理

1．bestblk

功能：

确定块操作的大小

语法：

```
siz= bestblk([m n],k)
[mb,nb]= bestblk([m n],k)
```

举例：

```
siz = bestblk([640 800],72)
```

参考： blkproc

2．blkproc

功能：

实现图像的块显示

语法：

```
B= blkproc(A,[m n],fun)
B= blkproc(A,[m,n],fun,P1,P2,…)
B= blkproc(A,[m,n],[mborder nborder],fun,…)
B= blkproc(A,'indexed',…)
```

举例：

```
I = imread('cameraman.tif');
fun = @dct2;
J = blkproc(I,[8 8],fun);
imagesc(J), colormap(hot)
```

参考： colfilt, nlfilter, inline

3．col2im

功能：

将矩阵的列重新组织到块中

语法：

```
B= col2im(A,[m n],block_type,fun)
B= col2im(A,[m n],block_type,fun.P1,P2,…)
B= col2im(A,[m n],[mblock nblock],block_type,,fun,…)
B= col2im(A,'indexed',…)
```

举例：

无

参考： blkproc, colfilt, im2col, nlfilter

4．im2col

功能：

重调图像块为列

语法：

```
B= im2col(A,[m n],block_type)
B= im2col(A,[m n])
B= im2col(A,'indexed',…)
```

举例：

无

参考： blkproc, col2im, colfilt, nlfilter

5．nlfilter

功能：

进行边沿操作

语法：

```
B= nlfilter(A,[m n],fun)
B= nlfilter(A,[m n],fun,P1,P2,…)
B= nlfilter(A,'indexed',…)
```

举例：

```
B = nlfilter(A,[3 3],@myfun);
```

参考： blkproc, colfilt

二进制图像操作

1．applylut

功能：

在二进制图像中利用查表进行边沿操作

语法：

```
A= applylut(BW,lut)
```

举例：

```
lut = makelut('sum(x(:)) == 4',2);
BW1 = imread('text.tif');
BW2 = applylut(BW1,lut);
imshow(BW1)
figure, imshow(BW2)
```

参考： makelut

2．bwarea

功能：

计算二进制图像对象的面积

语法：

```
total= bwarea(BW)
```

举例：

```
BW = imread('circles.tif');
imshow(BW),
```

```
bwarea(BW)
```

参考：bweuler, bwperim

3．bweuler

功能：

计算二进制图像的欧拉数

语法：

```
eul= bweuler(BW,n)
```

举例：

```
BW = imread('circles.tif');
imshow(BW);
bweuler(BW)
```

参考：bwmorph, bwperim

4．bwfill

功能：

填充二进制图像的背景色

语法：

```
BW2= bwfill(BW1,c,r,n)
BW2= bwfill(BW1,n)
[BW2,idx]= bwfill(…)
BW2= bwfill(x,y,BW1,xi,yi,n)
[x,y,BW2,idx,xi,yi]= bwfill(…)
BW2= bwfill(BW1,'holes',n)
[BW2,idx]= bwfill(BW1,'holes',n)
```

举例：

```
BW1 = [1    0    0    0    0    0    0    0
       1    1    1    1    1    0    0    0
       1    0    0    0    1    0    1    0
       1    0    0    0    1    1    1    0
       1    1    1    1    0    1    1    1
       1    0    0    1    1    0    1    0
       1    0    0    0    1    0    1    0
       1    0    0    0    1    1    1    0]
BW2 = bwfill(BW1,3,3,8)
```

参考：bwselect, roifill

5．bwlabel

功能：

标注二进制图像中连接的部分

语法：

```
L= bwlabel(BW,n)
[L,num]= bwlabel(BW,n)
```

举例：

```
BW = [1  1  1  0  0  0  0  0
      1  1  1  0  1  1  0  0
      1  1  1  0  1  1  0  0
      1  1  1  0  0  0  1  0
      1  1  1  0  0  0  1  0
      1  1  1  0  0  0  1  0
      1  1  1  0  0  1  1  0
      1  1  1  0  0  0  0  0];
L = bwlabel(BW,4)
```

参考：bweuler, bwlabeln, bwselect, label2rgb

6．bwmorph

功能：

提取二进制图像中的轮廓

语法：

```
BW2= bwmorph(BW1,operation)
BW2= bwmorph(BW1,operation,n)
```

举例：

```
BW1 = imread('circles.tif');
imshow(BW1);
BW2 = bwmorph(BW1,'remove');
BW3 = bwmorph(BW1,'skel',Inf);
imshow(BW2)
figure, imshow(BW3)
```

参考：bweuler, bwperim, dilate, erode

7．bwperim

功能：

计算二进制图像中对象的周长

语法：

```
BW2= bwperim(BW1,n)
```

举例：

```
BW1 = imread('circbw.tif');
BW2 = bwperim(BW1,8),
imshow(BW1)
figure, imshow(BW2)
```

参考：bwarea, bweuler, bwfill, conndef

8．bwselect

功能：

在二进制图像中选择图像

语法：

```
BW2= bwselect(BW1,c,r,n)
BW2= bwselect(BW1,n)
[BW2,idx]= bwselect(…)
```

举例:

```
BW1 = imread('text.tif');
c = [16 90 144];
r = [85 197 247];
BW2 = bwselect(BW1,c,r,4);
imshow(BW1)
figure, imshow(BW2)
```

参考：bwfill, bwlabel, impixel, roipoly, roifill

9．dilate

功能:

放大二进制图像

语法:

```
BW2= dilate(BW1,SE)
BW2= dilate(BW1,SE,alg)
BW2= dilate(BW1,SE,…,n)
```

举例:

```
BW1 = imread('text.tif');
SE = ones(6,2);
BW2 = dilate(BW1,SE);
imshow(BW1)
figure, imshow(BW2)
```

参考：bwmorph, imdilate, imerode

10．erode

功能:

弱化二进制图像的边界

语法:

```
BW2= erode (BW1,SE)
BW2= erode (BW1,SE,alg)
BW2= erode (BW1,SE,…,n)
```

举例:

```
BW1 = imread('text.tif');
SE = ones(3,1);
BW2 = erode(BW1,SE);
imshow(BW1)
figure, imshow(BW2)
```

参考：bwmorph, imdilate, imerode

11．makelut

功能:

创建一个用于 applylut 的查询表

语法:

```
lut=makelut(fun,n)
```

```
lut=makelut(fun,n,P1,P2,…)
```

举例：

```
f = inline('sum(x(:)) >= 2');
lut = makelut(f,2)
```

参考： applylut

区域处理

1. roicolor

功能：

选择感兴趣的颜色区

语法：

```
BW= roicolor (A,low,high)
BW2= roicolor (A,v)
```

举例：

```
I = imread('rice.tif');
BW = roicolor(I,128,255);
imshow(I);
figure, imshow(BW)
```

参考： roifilt2, roipoly

2. roifill

功能：

在图像的任意区域中进行平滑插补

语法：

```
J= roifill(I,c,r)
J= roifill(I)
J= roifill(I,BW)
[J,BW]= roifill(…)
J= roifill(x,y,I,xi,yi)
[x,y,J,BW,xi,yi]= roifill(…)
```

举例：

```
I = imread('eight.tif');
c = [222 272 300 270 221 194];
r = [21 21 75 121 121 75];
J = roifill(I,c,r);
imshow(I)
figure, imshow(J)
```

参考： roifilt2, roipoly

3. roifilt2

功能：

过滤敏感区域

语法：

```
J= roifilt2(h,I,BW)
J= roifilt2(I,BW,fun)
J= roifilt2(I,BW,fun,P1,P2,…)
```

举例：

```
I = imread('eight.tif');
c = [222 272 300 270 221 194];
r = [21 21 75 121 121 75];
BW = roipoly(I,c,r);
h = fspecial('unsharp');
J = roifilt2(h,I,BW);
imshow(J), figure, imshow(J)
```

参考：filter2, roipoly

4．roipoly

功能：

选择一个敏感的多边形区域

语法：

```
BW= roipoly (I,c,r)
BW= roipoly (I)
BW= roipoly (x,y,I,xi,yi)
[BW,xi,yi]= roipoly(…)
[x,y,BW,xi,yi]= roipoly(…)
```

举例：

```
I = imread('eight.tif');
c = [222 272 300 270 221 194];
r = [21 21 75 121 121 75];
BW = roipoly(I,c,r);
imshow(I)
figure, imshow(BW)
```

参考：roifilt2, roicolor, roifill

颜色映像处理

1．brighten

功能：

增加或降低颜色映射表的亮度

语法：

```
brighten(beta)
newmap= brighten(beta)
newmap= brighten(map,beta)
brighten(fig,beta)
```

举例：

```
beta = .5; brighten(beta);
beta = --.5; brighten(beta);
```

参考：colormap, rgbplot

2．cmpermute

功能：

调整颜色映像表中的颜色

语法：

```
[Y,newmap]= cmpermute(X,map)
[Y,newmap]= cmpermute(X,map,index)
```

举例：

```
ntsc = rgb2ntsc(map);
[dum,index] = sort(ntsc(:,1));
[Y,newmap] = cmpermute(X,map,index);
```

参考：randperm, sort

3．cmunique

功能：

查找颜色映像表中特定的颜色和相应的图像

语法：

```
[Y,newmap]= cmunique(X,map)
[Y,newmap]= cmunique(RGB)
[Y,newmap]= cmunique(I)
```

举例：

无

参考：gray2ind, rgb2ind

4．imapprox

功能：

对索引图像进行近似处理

语法：

```
[Y,newmap]= imapprox(X,map,n)
[Y,newmap]= imapprox(X,map,tol)
Y= imapprox(X,map,newmap)
[…]= imapprox(…,dither_option)
```

举例：

无

参考：cmunique, dither, rgb2ind

5．rgbplot

功能：

划分颜色映像表

语法：

```
rgbplot(map)
```

举例:

rgbplot(copper)

参考: colormap

颜色空间转换

1．hsv2rgb

功能:

转换 HSV 值为 RGB 颜色空间

语法:

```
rgbmap= hsv2rgb(hsvmap)
RGB= hsv2rgb(HSV)
```

举例:

无

参考: brighten, colormap, rgb2hsv

2．ntsc2rgb

功能:

转换 NTSC 值为 RGB 颜色空间

语法:

```
rgbmap= ntsc2rgb(yiqmap)
RGB= hsv2rgb(YIQ)
```

举例:

无

参考: rgb2ntsc, rgb2ind, ind2rgb, ind2gray

3．rgb2hsv

功能:

转换 RGB 值为 HSV 颜色空间

语法:

```
hsvmap= rgb2hsv (rgbmap)
HSV = rgb2hsv (RGB)
```

举例:

无

参考: brighten, colormap, hsv2rgb,rgbplot

4．rgb2ntsc

功能:

转换 RGB 值为 NTSC 颜色空间

语法:

```
yiqmap= rgbsntsc(rgbvmap)
YIQ= rgb2ntsc(RGB)
```

举例：

无

参考：ntsc2rgb, rgb2ind, ind2rgb, ind2gray

5．rgb2ycbcr

功能：

转换 RGB 值为 YCbCr 颜色空间

语法：

```
ycbcrmap= rgb2ycbcr(rgbmap)
YCBER=rgb2ycbcr(RGB)
```

举例：

无

参考：ntsc2rgb, rgb2ntsc, ycbcr2rgb

6．ycbcr2rgb

功能：

转换 YCbCr 值为 RGB 颜色空间

语法：

```
rgbmap=ycbcr2rgb(ycbmap)
RGB=ycbcr2rgb(YCBCR)
```

举例：

无

参考：ntsc2rgb, rgb2ntsc, rgb2ycbcr

图像类型和类型转换

1．dither

功能：

通过抖动增加外观颜色分辨率

语法：

```
X= dither(RGB,map)
BW= dither(I)
```

举例：

无

参考：ntsc2rgb, rgb2ntsc, rgb2ycbcr

2．gray2ind

功能：

转换灰度图像为索引图像

语法：

```
[X,map]= gray2ind(I,n)
```

举例：

无

参考：ind2gray

3．grayslice

功能：

从灰度图像创建索引图像

语法：

```
X= grayslice(I,n)
X= grayslice(I,v)
```

举例：

```
I = imread('ngc4024m.tif');
X = grayslice(I,16);
imshow(I)
figure, imshow(X,jet(16))
```

参考：gray2ind

4．im2bw

功能：

转换图像为二进制图像

语法：

```
BW= im2bw(I,level)
BW= im2bw(X,map,level)
BW= im2bw(RGB,level)
```

举例：

```
load trees
BW = im2bw(X,map,0.4);
imshow(X,map)
figure, imshow(BW)
```

参考：graythresh, ind2gray, rgb2gray

5．im2double

功能：

转换图像矩阵为双精度型

语法：

```
I2= im2double(I1)
RGB2= im2double(RGB1)
BW2= im2double(BW1)
X2= im2double(X1,'indexed')
```

举例：

无

参考：double, im2uint8, uint8



6. im2unit8

功能：

转换图像矩阵为 8 位无符号数型

语法：

```
I2= im2unit8(I1)
RGB2= im2unit8(RGB1)
BW2= im2unit8(BW1)
X2= im2unit8(X1,'indexed')
```

举例：

无

参考： im2uint16, double, im2double, uint8, imapprox, uint16

7. im2unit16

功能：

转换图像矩阵为 16 位无符号数型

语法：

```
I2= im2unit16(I1)
RGB2= im2unit16(RGB1)
X2= im2unit16(X1,'indexed')
```

举例：

无

参考： im2uint8, double, im2double, uint8, uint16, imapprox

8. ind2gray

功能：

将检索图像转换为灰度图像

语法：

```
I= ind2gray(X,map)
```

举例：

```
load trees
I = ind2gray(X,map);
imshow(X,map)
figure,imshow(I)
```

参考： gray2ind, imshow, rgb2ntsc

9. int2rgb

功能：

将检索图像转换为 RGB 图像

语法：

```
RGB= int2rgb(X,map)
```

举例：

无

参考：ind2gray, rgb2ind

10．isbw

功能：

判断是否为二进制图像

语法：

```
flag= isbw(A)
```

举例：

无

参考：isind, isgray, isrgb

11．isgray

功能：

判断是否为灰度图像

语法：

```
flag=isgray(A)
```

举例：

无

参考：isbw, isind, isrgb

12．isind

功能：

判断是否为索引图像

语法：

```
flag=isind(A)
```

举例：

无

参考：isbw, isgray, isrgb

13．isgrb

功能：

判断是否为 RGB 图像

语法：

```
flag=isrgb(A)
```

举例：

无

参考：isbw, isgray, isind

14．mat2gray

功能：

转换矩阵为灰度图像

语法：

```
I= mat2gray(A,[amin amax])
I= mat2gray(A)
```

举例：

```
I = imread('rice.tif');
J = filter2(fspecial('sobel'),I);
K = mat2gray(J);
imshow(I)
figure, imshow(K)
```

参考： gray2ind

15、rgb2gray

功能：

转换 RGB 图像或颜色映像表为灰度图像

语法：

```
I= rgb2gray(RGB)
Newmap= rgb2gray(map)
```

举例：

无

参考： ind2gray, ntsc2rgb, rgb2ind, rgb2ntsc

16、rgb2ind

功能：

转换 RGB 图像为索引图像

语法：

```
[X,map]= rgb2ind(RGB,tol)
[X,map]= rgb2ind(RGB,n)
X= rgb2ind(RGB,map)
[…]= rgb2ind(…,dither_option)
```

举例：

```
RGB = imread('flowers.tif');
[X,map] = rgb2ind(RGB,128);
imshow(X,map)
```

参考： cmunique, dither, imapprox, ind2rgb, rgb2gray

附录 B　MATLAB 6.5 其他新特性

B.1　Simulink 5.0 新特性

Simulink 是一种框图模型建模环境，用于动态系统的仿真、性能评估，并进行控制系统、DSP 系统和通信系统的设计。该软件在 GUI 和运行引擎方面的改进有：

- 图形化的程序调试器，使错误分析及调试操作简单起来。
- 支持大多数 Simulink 模块间以矩阵形式交换数据。
- 支持更高速的基于帧信号处理的 DSP 应用。
- 支持大模型开发的新特性，包括更新的工具栏选项和增强的行编辑功能。
- 集成的查找对话框工具方便了元件在 Simulink 模型和（或）状态流图中的搜索。
- 增加了 Simulink 数据对象，支持用户自定义数据结构。
- 新的航空库及示例。
- 提高了库浏览器和模型浏览器的使用性能。
- 废除了库连接，以方便库模型块的编辑。
- 增强了的可配置子系统，便于设计模型向实际实现转化。
- 具有筛选算法的高级查表模块，使仿真和代码生成更加快速、精确、灵活。
- 单一视窗模式节省了宝贵的屏幕空间。

B.2　MathWorks Release 13 新产品

1．**航空工具箱**（Aerospace Blockset）

航空工具箱以仿真为基础，提供具体的工具用于飞机、飞船、导弹及推进系统和子系统的建模、集成和仿真：

- 对于宇航式飞行器，提供了推进、控制系统、系统动力学及激励控制箱。
- 用航空工具箱的组件提供了六自由度和三自由度的实例模型。
- 包含重力、空气和风的环境模型。
- 使用空间表示转换模块增强了坐标转换功能。
- 能实现物理特性的单元转换。
- 使用 Handle Graphics 技术，提供了动画模块，实现三自由度和六自由度的动画演示。

2．**曲线拟合工具箱**（Curve Fitting Toolbox）

曲线拟合工具箱提供了数据预处理的各种方法，比如生成、比较、分析和管理模型。

所有的功能既可以用命令行来实现，也可以用图形用户界面来实现：

- 预处理方法，包括数据的标定、截断、滤波和奇异值的去除等。
- 扩展了线性和非线性参数拟合模型库。对于非线性模型，起始点和参数的解算程序都进行了优化。
- 有各种各样的参数和非参数模型，包括最小二乘法、加权最小二乘法和鲁棒拟合过程（包括有参数边界限制和没有参数边界限制）。
- 自定义参数和非参数模型扩展。
- • 用样条和内插法实现非参数拟合。
- 拟合数据的插补、外推、微分和积分。

3．Motorola MPC555 嵌入式目标（Embedded Target for Motorola MPC555）

Motorola MPC555 嵌入式目标可以将 MATLAB 实时工作空间编码器生成的代码直接配置到 MPC555 微控制器。Motorola MPC555 嵌入式目标依靠 MATLAB 实时工作空间编码器来为 Motorola MPC555 生成和提供具体的代码：

- 为 MPC555 提供快速原型和产品代码生成。
- 针对 Motorola MPC555 MIOS，QADC 和 TouCAN 的设备驱动块。
- 通过板载调试器（onboard debugger）或 CAN（controller area network）下载应用到 Motorola MPC555。
- 通过处理器在回路联合仿真的方式验证算法代码。
- 生成详细的 HTML 代码生成报告，包括 RAM/ROM 信息。

4．C6000™ DSP 平台嵌入式目标（Embedded Target for C6000™ DSP Platform）

简化 Texas Instruments DSP 软件设计和分析流程，直接从 MathWorks 环境生成高效代码。你可以使用 Simulink，DSP Blockset 和 Communications Blockset 中的块开发层次化的 DSP 算法框图模型，然后通过 Real-Time Workshop 生成无歧义的可执行算法，并可以进一步由 DSP 软件工程师优化：

- 自动生成支持 Texas Instruments DSP 目标的代码。
- 无需 DSP 编程，就可以实时验证算法。
- 生成 Code Composer Studio™ 项目格式的带注解的 C 代码。

5．MATLAB COM Builder

MATLAB COM Builder 可以很容易地将 MATLAB 算法转化为 COM 对象，可以用于任何基于 COM 的应用：

- 通过简便易用的 GUI 将 MATLAB 算法转换成 COM 对象。
- 生成 COM 对象，可以供 Visual Basic、C/C++、 Microsoft Excel 和其他任何基于 COM 的应用使用。
- 允许修改和查看 MATLAB 生成的代码，可以更好地理解你所引用的函数。
- 通过 MATLAB 生成的 COM 对象可以自由发布。

6．MATLAB Excel Builder

MATLAB Excel Builder 可以很容易地将复杂的 MATLAB 算法转化为独立的 Excel 插



件（add-ins），用户可以利用以矩阵为基础、灵活的 MATLAB 编程环境及拥有大量可用的数学和图像函数的优点，从而迅速建立计算更深入的模型：

- 通过简便易用的 GUI 将 MATLAB 算法转换成 Excel 插件（add-in）。
- 自动生成 .dll 和 Visual Basic 应用文件，并且可以输入到 Excel 中。
- 生成的插件函数比 Visual Basic for Applications（VBA）生成的快 95%。
- 允许修改和查看 MATLAB 生成的代码，可以对你所引用的函数背后的逻辑有更好的理解。

7．Code Composer Studio **开发工具的** MATLAB **连接**

简化 Texas Instruments DSP 软件设计和分析流程，实现在 TI 软件开发环境，实时 DSP 硬件和 MATLAB 之间的通信。算法开发人员、系统设计人员和嵌入工程师通过此工具可以在 MATLAB 中测试、验证和可视化 DSP 软件，消除了 DSP 算法研究和实现之间的缺口：

提供 M 文件功能，允许你对数据传输和验证任务进行定制和自动化。

- 在 MATLAB 和 Texas Instruments DSP 设备之间提供数据传输能力，而不必停止目标应用上的正在执行的任务。
- 硬件对象和 Code Composer Studio 帮助测试、验证和可视化 DSP 代码。

8．**基于模型校准工具箱**（Model-Based Calibration Toolbox）

基于模型校准工具箱提供设计工具来校准功率系统。它是建立在 MATLAB 高性能的计算环境及仿真能力的基础上，减少了功率器的检测时间，提高了工程效率，节省了校准时间，因而有潜力改进功率系统性能和可靠性：

- 新的经典设计：Plackett-Burman 两水平筛选设计（tow-level screening design），规则单纯形（Regular Simplex）—— 一个效率高的一阶设计方法。
- 使用更方便，可以将详细用户信息存储起来，可以在 Model Browser 中跟踪项目文件的历史列表。
- 新的上下文菜单，可以复制、删除和重命名模型。
- 在现有的所有模型绘图放大功能之上，增加了新的数据绘图放大工具。
- 在 CAGE 用户界面中浏览大型数据集，速度可以提高 40% 以上。
- 支持 Simulink 库中的高级查表类型。

9．**机械仿真**（SimMechanics）

它允许工程师在 Simulink 环境中仿真机械系统。应用 SimMechanics 可以直接建立机械部件的模型，仿真它们的运动及分析结果，而不需要推导数学方程：

- 增强的非线性方程求解器，仿真速度更快。
- 优化的模型线性化。
- 新的邻接（adjoining）特性，方便对相邻的其他物体的坐标系位置和方向的定义。
- 对于并联机械模型，用户可以选择和观察切断铰链（cut joints）。
- 图形用户界面增强，模型动画功能增强。

具体内容读者可到网站 http://www.mathworks.com 查阅。

附录C　MATLAB 6.5 安装问题指南

C.1　MATLAB 6.5 为什么安装后不能启动

经常有朋友遇到这个问题，其实在 Windows 98 或其第 2 版下安装 MATLAB 6.5 应该没有问题（也有部分可能会在安装了 IE 5.5 或更高版本、更新了 Windows 的一些字库以后出现），而在 Windows Me 或 Windows 2000 下安装 MATLAB 6.5 不能启动的主要原因是它的部分中文字库和操作系统的字库重名造成冲突，下面给出几种可行的解决方法以供选择。

1．更改区域设置

在控制面板里有区域设置一项，一般中文的 Windows 操作系统上默认的区域设置是“中国（中文）”，如图 C-1 所示。这里我们只要把它更改为“英语（美国）”之后重新启动计算机，MATLAB 6.5 就可以正常地运行了。不用担心，改过区域设置之后，操作系统仍然是中文操作系统，并不会给我们的使用带来太多的麻烦（当然有些小的显示等方面的问题）。这也是最省事的一种解决方法了。



图 C-1　更改区域设置

2．消除重名字库

可以直接在操作系统中将重名字库给删除，或是在 MATLAB 6.5 的字体配置文件中改

变更名字库相关信息。前一种删除方法将导致该字库从操作系统中被完全删掉，其他应用软件将无法使用该字库，因此并不十分理想，不过却也非常方便。后一种方法，该字库没有从操作系统中删掉，只是在 MATLAB 6.5 的字体属性配置文件中改变了相关信息，因此并不影响其他应用软件使用这些字库。这两种删除方法共同的最核心的问题就在于要知道是哪一些字库重名了，而这往往需要去试验或是获得他人的经验。目前在一般的情况下，大约只有"新宋体"这个字库有重名的问题。第一种删除字体的方法也非常简单（在字体目录下将相应的文件删除即可），但一般不提倡使用。这里，我们就介绍如何更改 MATLAB 6.5 的字体配置文件。

首先，MATLAB 6.5 这个导致死机的字体属性配置文件是 font.properties.zh，这个文件在目录$MATLAB\sys\java\jre\win32\jre\lib\下（$MATLAB 表示 MATLAB 6.5 的安装目录）。我们可以用一个文本编辑器来编辑这个文件。

在 $MATLAB\sys\java\jre\win32\jre\lib\font.properties.zh 文件的最后加上：

```
# Font entry added by The MathWorks:"新宋体"
tmw36525210.0=\u65b0\u5b8b\u4f53,ANSI_CHARSET
tmw36525210.1=\u65b0\u5b8b\u4f53,GB2312_CHARSET
tmw36525210.2=WingDings,SYMBOL_CHARSET,NEED_CONVERTED
tmw36525210.3=Symbol,SYMBOL_CHARSET,NEED_CONVERTED
fontcharset.tmw36525210.2=sun.awt.windows.CharToByteWingDings
fontcharset.tmw36525210.3=sun.awt.CharToByteSymbol

# Font entry added by The MathWorks:"@新宋体"
tmw39767002.0=@\u65b0\u5b8b\u4f53,ANSI_CHARSET
tmw39767002.1=@\u65b0\u5b8b\u4f53,GB2312_CHARSET
tmw39767002.2=WingDings,SYMBOL_CHARSET,NEED_CONVERTED
tmw39767002.3=Symbol,SYMBOL_CHARSET,NEED_CONVERTED
fontcharset.tmw39767002.2=sun.awt.windows.CharToByteWingDings
fontcharset.tmw39767002.3=sun.awt.CharToByteSymbol
```

以上结果为在 Windows 2000 Professional 简体中文版上所得，如果你的系统中还有其他有问题的中文字库，需要同样的处理。经过检验可以找到一些没有问题的中文字库如宋体、黑体、仿宋、楷体、幼圆、隶书。

这两种手段都比较麻烦，需要一些经验和试验，如果不小心还可能会出一些问题，下面我们就介绍 MathWorks 公司的解决方案。

3．使用 MathWorks 公司提供的技术支援

经过 MathWorks 公司技术人员的研究，终于找到了问题的根源——Java 虚拟机用来将标准 Java 字体转换为系统字体所用的映射属性文件 font.properties.zh 及其映射程序中存在一个小 BUG。MathWorks 公司已经将补丁文件 mwt.jar 发布在其公司的网站 http://www.mathworks.com 上了，该文件最新的下载位置是 ftp://ftp.mathworks.com/pub/tech-support/solutions/s26990。下载了该文件后用其替换目录$MATLAB\java\jar\下的同名文件（当然最好对初始的文件做一个备份）。然后，重新启动 MATLAB 6.5，问题就可以解决了。如果仍有一些其他问题，可以到该公司或其他一些相关网站上查询。

C.2　安装时的更新 Java 虚拟机的问题

这个问题一般比较偶然地出现在 Windows 98 和 Windows NT 环境中，有的时候因为安装其他软件时已经解决了这个问题，在安装 MATLAB 6.5 时就会比较顺利。但偶尔会出现要求更新 Java 虚拟机的提示消息框，这时我们应当更新我们的 Java 环境。其软件一般可以在 MATLAB 6.5 的安装盘中找到，如果是下载的软件，可以再到网址 http://www.microsoft.com/java/vm/dl_vm40.htm 去下载相应的软件 msvm.exe（对应于 Windows 98 和 Windows NT 操作系统）。如果是 Windows 2000 操作系统，只需要安装升级补丁 SP2。

C.3　PDF 文档的获取

国内的 MATLAB 6.5 软件大多是下载的软件，因此帮助文件不全。下面的这两个网址可以给大家一点帮助。

国外：http://www.mathworks.com/access/helpdesk/help/helpdesk.shtml

国内：http://science.fire.ustc.edu.cn/MATLAB6.html

参 考 文 献

[1] 陈桂明，张明照，戚红鱼. 应用 MATLAB 语言处理数字信号与数字图像. 北京：科学出版社，2000

[2] 章毓晋. 图像处理和分析. 北京：清华大学出版社，1999

[3] 周新伦，柳健. 数字图像处理. 北京：国防工业出版社，1986

[4] The MathWorks, Inc. MATLAB 6.5 帮助文档

[5] 孙家广，杨长贵. 计算机图形学. 北京：清华大学出版社，1995

[6] 杜跃鹏，原新生. Matlab 在分形几何中的应用. 平顶山师专学报：vol16(4)，2001.11

[7] 刘斌. 图像编码的新方法——分形几何. 南昌大学学报：vol17(3)，1997.9

[8] The MathWorks, Inc. http://www.mathworks.com,2002